Editorial

Ya se pasaron seis años de la edición del primer número de esta revista y con la obstinación de seguir adelante queriendo cumplir nuestros objetivos iniciales de crear una comunidad académica Colombiana, éstos han sido cumplidos parcialmente. La revista ya ha sido indexada en DBLP (http://www.informatik.uni-trier.de/~ley/db/journals/rcc/index.html), uno de los índices más reconocidos a nivel mundial en el área de computación, y en Publindex y Latindex. Además, existe una corriente muy fuerte de crear una Sociedad Colombiana de Computación que pretende ser un foro de discusión académica en el área. Invitamos a todos los académicos Colombianos, o no, en Colombia y en el exterior, para que se unan a nosotros apoyándonos enviándonos trabajos o como evaluadores. Este trabajo no ha sido fácil, precisamente, por la búsqueda de buenos evaluadores que nos apoyen en el mantenimiento de la calidad de nuestra revista.

En esta edición tenemos una selección internacional, incluyendo artículos de países tales como Francia, Argentina, España, Inglaterra y por supuesto Colombia.

En su articulo "Evaluación de Herramientas de Extracción Automática de Conceptos dentro de un Ambiente de Biblioteca Digital", Abascal y Rumpler aprovechan el uso de "metadatos" para el mejoramiento de las consultas dentro de una biblioteca digital, para proponer un enfoque de "metadados" que permitan describir, en su caso, las tesis doctorales de una biblioteca digital. Ellos presentan una evaluación de varias herramientas automáticas de extracción de estos "metadatos" (conceptos), además de un prototipo de herramienta de anotación desarrollado por ellos para insertar de manera automática conceptos a las tesis digitales.

En el artículo "Sliding Mode Controller for Robust Force Control of Hydraulic Actuator with Environmental Uncertainties", Boumhidi y Mrabti seleccionan un modelo reducido de orden lineal ("Sliding Mode Controller") para describir el servo-activador hidráulico con grandes incertidumbres ambientales.

En el artículo "Ontology-Based Data Integration Methods: A Framework for Comparison", Buccella, Cechich y Brisaboa, presentan una revisión de siete sistemas y tres propuestas para la integración de datos basada en Ontologías. Los autores muestran semejanzas y diferencias de los sistemas y ofrecen un esquema para su comparación y descripción.

En el artículo "Combining Symbolic Execution and Model Checking to Reduce Dynamic Program Analysis Overhead", Néstor Cataño integra las técnicas de chequeo de modelos ("model checking") y razonamiento simbólico cuando se realiza análisis dinámico de programas.

En el artículo "Algoritmo de Emparejamiento de Perfiles en Servicios Web Semánticos" Samper, Carrillo y Martínez presentan las principales características de un algoritmo de emparejamiento de servicios Web semánticos y describen los principales componentes

relacionados con el proceso de implementación; el algoritmo utiliza las capacidades proporcionadas por la ontología de descripción de servicios y su implementación interactúa con ontologías de descripciones de conceptos desarrolladas en DAML+OIL y descripciones de servicios en DAML-S, usando como repositorio / razonador el sistema Sesame+BOR.

Como pueden ver los artículos tratan de temas bastante actuales e importantes dentro del área de computación, esperamos que los disfruten.

Bucaramanga, junio de 2005

Los editores
Alvaro Enrique Arenas y José de Jesús Pérez

Evaluación de Herramientas de Extracción Automática de Conceptos Dentro de un Ambiente de Biblioteca Digital

Rocío Abascal [1] Béatrice Rumpler[1]

Resumen

El rápido avance de la tecnología ha originado la proliferación de fuentes de información digital. Esta evolución informática ha provocado la creación de bibliotecas digitales que han ido convirtiendose poco a poco en un gran pilar para la difusión del conocimiento. Sin embargo, la información contenida en las bibliotecas digitales aún no está descrita totalmente y su explotación es aún insuficiente. Recientemente, se ha comprobado que la descripción de la información usando *"metadatos"* puede ser primordial para el mejoramiento de la consulta de la información dentro de una biblioteca digital. Nuestro enfoque está basado en la creación e introducción de nuevos *"metadatos"* capaces de describir, en nuestro caso, las tesis doctorales de una biblioteca digital. Estos *"metadatos"* corresponden a los conceptos más importantes de cada una de las tesis. Actualmente, la identificación manual de conceptos es un largo proceso llevado a cabo por un especialista del área. Por lo tanto, es importante hacer uso de herramientas capaces de extraer automáticamente conceptos. En este artículo analizamos cuatro herramientas de PLN (Procesamiento del Lenguaje Natural) capaces de extraer automáticamente los conceptos claves de un corpus. Estas herramientas son: (1) TerminologyExtractor de Chamblon Systems Inc., (2) Xerox Terminology Suite de Xerox, (3) Nomino de Nomino Technologies y (4) Copernic Summarizer de NRC. Este artículo presenta también un prototipo de herramienta de anotación desarrollado para insertar de manera automática conceptos a las tesis digitales.

Palabras claves: *Biblioteca digital, metadatos, Procesamiento del Lenguaje Natural, extracción de información, anotación, búsqueda de información.*

Abstract

The fast advance of the technology has originated the proliferation of digital sources of information. This computer evolution has caused the creation of digital libraries that have become a big pillar for the diffusion of knowledge. However, the information contained in the digital libraries is not totally described and its exploitation is still insufficient. Recently, it has been proven that describing the information by using *"metadata"* can be fundamental for the improvement of the research of the information within a digital library. Our approach is based on the creation and the introduction of new *"metadata"* able to describe, in our case, the PhD theses of the digital library. These *"metadata"* correspond to the most important concepts of each one of the theses contained in the digital library. At the moment, manual identification of concepts is a long process that is carried out by a specialist of the area. Therefore, we considered the use of tools to be able to automatically extract concepts. In this article we analyze four tools of NLP (Natural Language Processing) able to automatically extract the key concepts of a corpus. These tools are: (1) TerminologyExtractor of Chamblon Systems Inc., (2) Xerox Terminology Suite of Xerox, (3) Nomino of Nomino Technologies and (4) Copernic Summarizer of NRC. This paper also presents a prototype developed to automatically insert concepts into digital theses.

Keywords: *Digital library, metadata, Natural Language Processing, information extraction,annotation, information research.*

[1] INSA de Lyon – LIRIS, 7 Avenue J. Capelle Bät 502 – Blaise Pascal F69621 Villeurbanne cedex FRANCE. {Rocio.Abascal, Béatrice.Rumpler}@insa-lyon.fr

1 Introducción

A pesar de que los sistemas de recuperación de información son cada día más sofisticados y precisos, las necesidades del usuario aún no están completamente cubiertas. Los usuarios todavía deben juzgar sobre la pertinencia de los documentos obtenidos al realizar una búsqueda de información. Algunos de los criterios utilizados todavía por los motores de búsqueda para evaluar si la información es pertinente o no, se basan solamente en el número de veces en que las palabras clave aparecen en los documentos. Sin embargo, los usuarios deben leer grandes fragmentos del documento o en el peor caso el documento por completo para saber si es pertinente o no a sus necesidades. Con el fin de atacar este problema, nuestro enfoque está basado en la descripción conceptual de los documentos ya que consideramos que es fundamental para poder poner al alcance la información pertinente a los usuarios.

Nuestro estudio se realiza dentro del proyecto llamado CITHER[2] (Consulta integral de tesis en red). CITHER, desde 1997, lleva a cabo la difusión, vía Internet, de las tesis doctorales del INSA de Lyon, Francia. La difusión de las tesis se realiza utilizando el formato PDF (Portable Document Format). La descripción de ciertos datos correspondientes a la tesis se realiza actualmente a través del uso del formato Dublin Core. A partir de estos metadatos el sistema se apoya para realizar búsquedas de información. Actualmente, el usuario puede efectuar una consulta utilizando palabras correspondientes al título, el nombre del autor, la fecha o el año de edición. Sin embargo, la mayor parte del tiempo el usuario no conoce estos datos y por lo tanto no obtiene resultados pertinentes al efectuar una consulta. CITHER permite obtener la tesis completa durante una consulta pero cuando la tesis no es pertinente sería necesario obtener previamente fragmentos que en el caso de que el usuario considerara pertinentes pudiera desplegar en forma completa. Es en éste enfoque en el que trabajamos actualmente: caracterizar las tesis por medio de nuevos metadatos [2] y apoyar la búsqueda de información pertinente usando una ontología del área [1] para ofrecerle al usuario los fragmentos pertinentes de cada tesis.

Con el objetivo de caracterizar mejor el contenido de las tesis, hemos decidido usar nuevos *"metadatos"* basados en los conceptos que describen cada una de las tesis. La extracción manual de los conceptos que caracterizan un documento es una tarea larga y complicada ya que se necesita tener conocimiento del área de especialidad. Cuando se tienen pocos documentos a evaluar tal vez la extracción manual sea una tarea factible, sin embargo, al evaluar grandes cantidades de documentos se necesita que el proceso se lleve a cabo de manera automática o semiautomática. En este caso, hablamos de un proceso semiautomático ya que es el experto el que evaluará la pertinencia de los conceptos extraídos por las herramientas.

Para poder seleccionar una herramienta de PLN (Procesamiento de Lenguaje Natural) capaz de extraer automáticamente los conceptos de un corpus dado como entrada, hemos decidido buscar herramientas que puedan satisfacer nuestras necesidades. En este artículo, presentamos una evaluación empírica de cuatro herramientas que extraen automáticamente conceptos. Estas herramientas son: (1) TerminologyExtractor de Chamblon Systems Inc., (2) Xerox Terminology Suite de Xerox, (3) Nomino de Nomino Technologies y (4) Copernic Summarizer de NRC. La evaluación ha sido realizada mediante la comparación del grado de similitud entre los conceptos extraídos por cada una de las herramientas y una lista de referencia que contiene conceptos extraídos manualmente por un experto del área.

[2] http://docinsa.insa-lyon.fr/docinsa/these/

La estructura del artículo es la siguiente: en la Sección 2 presentamos un resumen de trabajos relacionados con el área de extracción automática de términos. En la Sección 3 presentamos las principales características de las herramientas evaluadas. El método utilizado para evaluar las diferentes herramientas lo presentamos en la Sección 4. La evaluación y los resultados de nuestro estudio son presentados en la Sección 5. En la Sección 6, presentamos un prototipo para una herramienta de anotación capaz de insertar los conceptos extraídos por las herramientas a cada una de las tesis de la biblioteca digital. Los conceptos extraídos corresponden a aquellos encontrados dentro de los capítulos más importantes de la tesis. Para esto, explicaremos rápidamente el análisis semántico que realizamos con el fin de encontrar una organización semántica dentro de las tesis y basarnos en esta a la hora de realizar la extracción de conceptos. Las conclusiones y el trabajo futuro se presentan al final del artículo.

2 Trabajo relacionado

Los términos son representaciones lingüísticas de los conceptos de un área en particular. Los términos pueden estar formados por una sola palabra, llamados así *"términos simples"* o también pueden estar formados por dos o más palabras, llamados así *"términos complejos"* [8]. Estos términos cuando son extraídos por una herramienta son llamados también *"candidatos-términos"*, ya que por lo general son términos complejos, es decir, grupos de palabras que pueden en cierto caso revelar cierto conocimiento tratado en un documento dado [12]. De esta manera, las herramientas que realizan una extracción automática de conceptos o términos, pueden ser aplicadas para la construcción de diccionarios especializados, mecanismos de traducción, indexación de libros y bibliotecas digitales, categorización de textos, etc. Además, el uso de conceptos para caracterizar documentos permite que estos conceptos sean usados para la extensión de peticiones de búsqueda y para facilitar al usuario el uso de un vocabulario adecuado. Otras de las muchas aplicaciones que podemos encontrar en el uso de conceptos es la capacidad para visualizar estos conceptos dentro del documento y hacer comparaciones de similitud entre documentos. Entre algunos de los trabajos que se orientan hacia la similitud de documentos basada en la extracción de conceptos podemos citar el trabajo de [9] que presenta un motor de búsqueda llamado Keyphind. Este motor permite clasificar los documentos de la colección utilizando los conceptos extraídos a través de la herramienta Kea [10].

La extracción de *"términos complejos"* está basada en algunas de las siguientes suposiciones:

(1) Los textos especializados contienen muchos términos que son siempre utilizados para referirse a los conocimientos o temas de un área especializada;

(2) Cuando un término es significativo tiene grandes posibilidades de que sea utilizado varias veces dentro del texto;

(3) La gran parte de los términos complejos están formados por más de una palabra, la cual raramente es utilizada sola;

(4) Los términos complejos están formados por un nombre o sustantivo seguido de un adjetivo; por ejemplo: *" biblioteca digital"* o *"inteligencia artificial"*.

Algunos algoritmos y métodos nuevos para la extracción de conceptos son presentados en [3,7,10]. En [6], el autor presenta un análisis de diferentes herramientas de extracción de conceptos a las analizadas en el presente artículo. En [12] el autor presenta una gráfica (basada en los resultados

una tesis de maestría[3] y de otros artículos) de comparación en términos de precisión entre las herramientas: LogiTerm, TermSearch, ATAO, Noun Phrase Search y Nomino.

Algunas técnicas utilizadas para determinar cuándo un término es complejo, están basadas en dos estrategias: (1) las *"estrategias lingüísticas"* y (2) las *"estrategias estadísticas o probabilísticas"*. Las *"estrategias lingüísticas"* utilizan conocimientos del idioma tratado, por ejemplo, basandose en ciertas categorías gramaticales. Las *"estrategias estadísticas o probabilísticas"* utilizan la frecuencia como método para la evaluación y la extracción de términos compuestos, basandose en que cuando un término es pertinente entonces significa que será utilizado varias veces dentro del documento. La herramienta *"Copernic Summarizer"* es evaluada en comparación a la herramienta *"IAI-extractor"* en [8]. Estas dos herramientas son un ejemplo de herramientas que usan la frecuencia para la extracción de términos.

Otro trabajo llevado a cabo en el área de extracción de conceptos incluye a las herramientas de generación automática de resúmenes. Estas herramientas no extraen automáticamente conceptos pero se nos hace interesante nombrarlas ya que algunas de ellas trabajan de manera muy similar a las herramientas que en este artículo presentamos. De esta forma entre las herramientas de generación automática de resúmenes podemos mencionar: TextSummary, HyperGen, WebSumm, DataHammer e IntelliScope, las cuales son evaluadas en términos de *"precisión"* y *"recuperación"* en [11].

3 Herramientas de extracción automática de conceptos

En esta sección, describimos las cuatro herramientas evaluadas, las cuales son: (1) Copernic Summarizer NRC, (2) Nomino de Nomino Technologies, (3) TerminologyExtractor de Chamblon Systems Inc., y (4) Xerox Terminology Suite de Xerox. Hemos evaluado únicamente las capacidades de cada una de las herramientas para efectuar la extracción automática de conceptos, generalmente llamada *"extracción o adquisición de términos"*.

3.1 Criterios para la selección de las herramientas

Los recientes avances en el área del PLN han fomentado la aparición de una nueva generación de herramientas que van más allá de la recuperación tradicional de la información, permitiendo a un sistema entender documentos de forma similar a como lo hacemos las personas, y por tanto posibilitando la extracción de conceptos semánticos útiles. Dentro de estas herramientas encontramos varios tipos de acuerdo a su funcionalidad: (1) análisis sintáctico, (2) extracción de conceptos, (3) etiquetado, (4) manejo del léxico, etc. En nuestro caso, para la selección de las herramientas buscamos únicamente aquellas que llevarán a cabo la extracción de conceptos.

Algunos de los criterios que utilizamos para dicha selección pueden resumirse en:

- *Tratamiento del idioma francés*: muchas de las herramientas que encontramos realizan la extracción automática de conceptos pero muy pocas están destinadas para analizar el idioma francés. Este criterio es de suma importancia ya que nuestro trabajo se realiza dentro de una biblioteca digital de tesis doctorales en francés.

[3] Love, S. Benchmarking the performance of two automated term-extraction Systems, Tesis de maestría, Montreal, Universidad de Montreal, 2000.

- *Facilidad de instalación*: todas las herramientas evaluadas se encuentran en linea y cuentan con una versión de evaluación.

- *Facilidad de uso*: nuestra evaluación tenía que llevarse a cabo en un corto plazo de tiempo por lo cual necesitábamos que todas las herramientas contarán con ejemplos y tutoriales que nos permitieran rápidamente dominar la herramienta.

- *Facilidad para contactar a los autores*: la gran mayoría de las herramientas analizadas, como Nomino, nos permitieron contar con la ayuda de los autores quienes nos mandaron más información al respecto y nos dieron una clave para usar Nomino ilimitadamente.

3.2 Copernic Summarizer

El algoritmo de extracción del NRC (National Research Council) es usado por Copernic Summarizer [5] para crear una lista de conceptos a partir de un documento dado. El principal objetivo de Copernic Summarizer es la creación automática de resúmenes [15], sin embargo, lo hemos evaluado sólo por su capacidad para efectuar también la extracción de conceptos. Esta capacidad permite extraer conceptos que están formados por más de una palabra. Al mismo tiempo que Copernic Summarizer crea un resumen para un documento, también produce una lista de conceptos y permite destacarlos dentro del resumen creado. Esta última capacidad, nos ha permitido poder revisar la importancia de los conceptos extraídos al poder visualizar los párrafos en los cuales aparecen dichos conceptos.

La versión evaluada de Copernic Summarizer corresponde a la 2.0 de Diciembre del 2001.

3.3 Nomino

Nomino es un motor de búsqueda desarrollado por la Universidad de Quebec en Montreal, anteriormente llamado Termino, y que actualmente es distribuido por Nomino Technologies [13]. Nomino adopta un enfoque morfosintáctico y es capaz de extraer términos pero también es capaz de identificar conceptos y de aislar sus relaciones semánticas [18]. El analizador morfológico usado por Nomino utiliza una *"lematización"* (*"stemming"* en inglés) lo que significa que el prefijo y el sufijo de una palabra son eliminados para obtener una palabra simple. Esta lematización permite reducir una palabra a su más mínima forma, llamada *"raíz"* [4]. Por ejemplo, si tenemos el concepto siguiente: *"bibliotecas digitales"*, Nomino formará la cadena siguiente: *"biblio digit"*. Así mismo, Nomino aplica ciertos criterios empíricos para filtrar el ruido encontrado en los conceptos extraídos. Estos criterios incluyen la frecuencia y la categoría, además del uso de unas listas que contienen palabras que no deben de ser tomadas en cuenta a la hora de efectuar una extracción. Nomino produce dos tipos de índices interactivos, los cuales contienen todos los conceptos que resumen de manera más precisa el contenido de un documento dado. Uno de los índices creados es muy general ya que contiene todas los conceptos encontrados en el documento. En cambio, el otro índice llamado *"índice prominente"* (en francés llamado: *"index de saillance"*) contiene los conceptos que para Nomino son los más pertinentes para describir un documento dado.

El llamado *"índice prominente"* está basado en dos principios: *"la ganancia para expresar"* y *"la ganancia para alcanzar"*. La *"ganancia para expresar"* clasifica los conceptos de acuerdo a su localización dentro del documento. Por ejemplo, si un párrafo habla sobre un sólo concepto entonces este concepto será clasificado como importante. La *"ganancia para alcanzar"* clasifica los conceptos de acuerdo a la frecuencia. De esta manera, si una palabra es muy rara entonces será

seleccionada como importante. Por ejemplo, en un documento dado, cuando se encuentran los conceptos *"biblioteca digital"* y *"arquitectura de la biblioteca digital"*, sólo el segundo concepto será seleccionado como pertinente ya que es mucho más completo que el primero. Sin embargo, los dos conceptos pueden ser seleccionados como pertinentes al mismo tiempo sólo en el caso de que el primer concepto *"biblioteca digital"* tenga una frecuencia mucho menor que el segundo concepto.

La versión evaluada de Nomino corresponde a la 4.2.22 del 25 de Julio del 2001. Esta versión de evaluación comprende todas las funciones de la herramienta y es posible utilizarla durante 10 horas no consecutivas.

3.4 TerminologyExtractor

TerminologyExtractor (TE) es una herramienta distribuida por Chamblon Systems Inc., [17]. El proceso de extracción de términos se lleva a cabo en dos pasos: (1) el paso de *"limpiar el texto"* y (2) el paso para la *"extracción de colocaciones o combinaciones"*.

El objetivo del paso de *"limpiar el texto"* es asegurarse que todas las inconsistencias sean eliminadas del texto de entrada. Para esto, TE utiliza una lista de exclusión que contiene todas las palabras que no son importantes a extraer. Además, TE utiliza un diccionario que es definido por el usuario y que contiene palabras importantes que no están contenidas en el diccionario del sistema.

El paso para la *"extracción de colocaciones o combinaciones"* produce una lista de colocaciones o términos con su respectiva frecuencia de aparición. TE también produce una lista de palabras y *"no-palabras"*. Estas *"no-palabras"* son todas aquellas palabras que no han sido encontradas en ambos diccionarios: el diccionario del sistema y el diccionario definido por el usuario.

La versión evaluada de TerminologyExtractor corresponde a la 3.0.

3.5 Xerox Terminology Suite

La herramienta Xerox Terminology Suite (XTS) de la compañía Xerox es un sistema de manejo de terminología que permite la creación de diccionarios multilingües y monolingües a partir de la extracción automática de términos o conceptos [19]. En nuestra evaluación, hemos considerado únicamente dos de los componentes de XTS: *"TermFinder"* y *"TermOrganizer"*.

El componente *"TermFinder"* construye automáticamente una base de datos de términos extraídos. Este componente utiliza herramientas lingüísticas para saber cómo una frase está construida. Así mismo, *"TermFinder"* utiliza una herramienta de extracción de sustantivos, por ejemplo, para saber cuando un sustantivo está seguido de un adjetivo calificativo.

El componente *"TermOrganizer"* se encarga del manejo de la base de datos que ha sido creada por el componente *"TermFinder"*. A través del uso de *"TermOrganizer"* es posible realizar modificaciones y especificaciones de los términos extraídos.

La versión evaluada de XTS corresponde a la 2.0 de Febrero del 2001. Esta versión corresponde a la herramienta llamada *"XTS the Terminology Suite"*.

En la Sección 4, presentamos la metodología utilizada para la evaluación de las herramientas de extracción automática de conceptos.

4 Metodología para la evaluación de herramientas de extracción automática de conceptos

La extracción de conceptos o términos está basada en dos diferentes enfoques: *"la asignación de frases claves"* y *"la extracción de frases claves"* [10]. Por el término *"frase clave"* nos referimos a una frase compuesta por dos o más palabras, las cuales describen en general algún tema o concepto tratado dentro de un documento en particular.

El trabajo de *"asignación de frases claves"* se lleva a cabo mediante el uso de un vocabulario controlado. De esta manera, se seleccionan los mejores conceptos o frases que describen un documento.

El trabajo de *"extracción de palabras claves"* se lleva a cabo mediante la selección de los conceptos que se encuentran dentro del documento mismo.

En este artículo presentamos sólo herramientas que llevan a cabo la *"extracción de palabras claves"*, lo cual significa que todos los conceptos que son extraídos siempre aparecen en el cuerpo del documento.

4.1 Evaluación de la eficiencia

Para evaluar la eficiencia de las herramientas de extracción de conceptos, decidimos comparar la lista generada de conceptos extraídos por las herramientas contra una lista de conceptos generada manualmente por un experto del área.

El primer paso para la evaluación de la eficiencia de las herramientas está basado en el número de coincidencias entre los conceptos generados por la herramienta y los conceptos generados manualmente por un experto del área. Un concepto es seleccionado como relevante sólo cuando existe una correspondencia entre la misma *"secuencia de radicales"*. Por *"secuencia de radicales"* nos referimos a una secuencia de palabras que aparecen en el mismo orden. Para esto, es importante la utilización de la *"lematización"*, la cual permite la búsqueda de las raíces de las palabras.

El segundo paso para la evaluación de la eficiencia de las herramientas está basado en la frecuencia con la que aparece un concepto dentro de un documento. Uno de los métodos más comunes utilizados para buscar la información pertinente está basado en la selección de aquellos conceptos que tienen una frecuencia alta de aparición. Muy a menudo ocurre que una palabra tiene una frecuencia alta incluso aún no siendo importante para el documento [14]. En este caso, para evitar este problema, en algunas herramientas como TerminologyExtractor la frecuencia alta de las palabras que no son pertinentes es eliminada utilizando una lista de palabras que contiene todas aquellas que no son importantes. Durante la fase de extracción, las palabras son comparadas a esta lista con el objetivo de seleccionar sólo aquellas que son relevantes.

4.2 Uso de las métricas de *"precisión"* y *"recuperación"* para clasificar la pertinencia de los términos extraídos

Para evaluar la lista de conceptos generada por cada herramienta, hemos comparado esta lista con una lista de conceptos generada manualmente por un experto del área. Las medidas usadas para la evaluación vienen del área de recuperación de información las cuales son: la *"precisión"* y la *"recuperación"*.

La *"precisión"* es una métrica utilizada para evaluar la capacidad que tiene el método de búsqueda para no obtener documentos no pertinentes. Es el cociente entre el total de documentos pertinentes obtenidos y el de documentos obtenidos. Por lo tanto, puede interpretarse como la probabilidad de que un documento obtenido sea pertinente [16]. En nuestro caso, la *"precisión"* es la proporción de SÓLO los conceptos relevantes recuperados.

La *"recuperación"* es una métrica utilizada para evaluar la capacidad que tiene el método de búsqueda para obtener documentos pertinentes, es decir, considerados útiles por quien hizo la consulta. Es el cociente entre el total de documentos pertinentes obtenidos y el de documentos pertinentes de la base. Por lo tanto, puede interpretarse como la probabilidad de que un documento pertinente sea obtenido [16]. En nuestro caso, la *"recuperación"* es la proporción de TODOS los conceptos recuperados, incluso aquellos recuperados que son irrelevantes.

Para evaluar los conceptos extraídos por las herramientas, es necesario primeramente clasificar los conceptos en la categoría de relevantes o irrelevantes. La evaluación llevada a cabo puede ser resumida en la siguiente tabla, Tabla 1.

	Concepto clasificado como pertinente por el humano	Concepto clasificado como no pertinente por el humano
Concepto clasificado como pertinente por la herramienta	*a*	*b*
Concepto clasificado como no pertinente por la herramienta	*c*	*d*

Tabla 1. Resumen del método de evaluación.

La variable "*a"* representa el número de conceptos generados por el humano que coinciden con los conceptos extraídos por la herramienta. Las variables "*b"* y "*c"* representan el número de veces o de conceptos en que el humano y la herramienta no concordaron en la fase de clasificación. La variable "*d"* representa el número de veces en que el humano y la herramienta concordaron en evaluar un concepto como irrelevante. De esta manera, las formulas de *"precisión"* y de *"recuperación"* son presentadas a continuación:

$$\text{Precisión} = \frac{a}{a+b}. \quad (1)$$

$$\text{Recuperación} = \frac{a}{a+c}. \quad (2)$$

En la siguiente sección, Sección 5, presentamos los pasos seguidos para efectuar la evaluación de las diferentes herramientas así como los resultados obtenidos

5 Resultados de la evaluación de las herramientas de extracción automática de conceptos

En esta sección, describimos la experimentación realizada y los resultados obtenidos en nuestra evaluación. Comenzamos con una descripción de los documentos utilizados como corpus y terminamos con un resumen de los resultados obtenidos.

5.1 El corpus

Los experimentos presentados en este artículo están basados en dos diferentes grupos de documentos. El primero corresponde a grandes documentos como lo son las tesis doctorales las cuales contienen más de 150 páginas cada una. El segundo grupo de documentos corresponde a pequeños artículos científicos compuestos de aproximadamente ocho páginas cada uno. Todo el corpus utilizado para la evaluación de las herramientas se encuentra en el idioma francés. Así mismo, tanto las tesis como los artículos científicos corresponden al área de la computación. Lo decidimos de esta manera ya que al analizar otras áreas seriamos incapaces de evaluar la pertinencia de los conceptos extraídos por las herramientas.

En la siguiente tabla, Tabla 2, presentamos el tamaño del corpus utilizado. Cabe recalcar, que en ciertos casos, al utilizar una versión de evaluación de las herramientas nos vimos confrontados al problema del tamaño del corpus. De esta forma, decidimos evaluar cada tesis por separado.

Nombre del corpus	Número total de documentos	Número total de palabras
Tesis doctorales	20	1 089 701
Artículos científicos	5	15 864

Tabla 2. Presentación del corpus utilizado para el análisis de las herramientas.

5.2 Diseño de experimentos

El primer paso para la evaluación de las herramientas consiste en la creación de una lista de referencia, la cual nos servirá como base para la comparación de la eficiencia de cada una de las herramientas. Esta lista de referencia contiene las palabras o conceptos claves dados por los autores de cada artículo. Esta primera lista es completada manualmente por un experto del área. La lista de referencia para el grupo de grandes documentos es creada en su totalidad ya que cada tesis no contiene palabras claves dadas por el autor de la misma. De esta forma, cada artículo y cada tesis tiene una lista de referencia con los conceptos que el experto considera que representan cada documento.

El segundo paso para la evaluación de las herramientas consiste en la comparación de los conceptos extraídos por cada herramienta contra los conceptos extraídos manualmente y que se encuentran en la lista de referencia. Un punto interesante en éste paso es la comparación de frases largas ya que éstas sólo son extraídas por algunas de las herramientas evaluadas. Por ejemplo, Nomino es la única herramienta capaz de extraer frases largas (conceptos) compuestas por más de tres palabras. Otro punto importante en éste paso es la visualización de algunos conceptos propuestos como pertinentes por las herramientas, por ejemplo en el caso de Copernic Summarizer que permite visualizar los párrafos donde aparecen dichos conceptos. Además, si la lista de referencias contiene un concepto como por ejemplo: *"ontología"*, es posible que a través de los conceptos extraídos por las herramientas se puedan encontrar conceptos que son mucho más expresivos y representativos, como*: "ontología de representación"* o *"conocimiento de la ontología"*.

El tercer paso es el análisis de los valores resultantes al aplicar las herramientas al corpus (Sección 5.1). Los valores que analizaremos en éste paso son:

- El número total de conceptos extraídos por la herramienta,
- El número total de conceptos extraídos por la herramienta y que aparecen en la lista de referencia creada manualmente,
- El número total de conceptos extraídos por la herramienta y que *no* aparecen en la lista de referencia creada manualmente,
- El número total de conceptos extraídos manualmente y que *no* aparecen en la lista generada por la herramienta.

5.2.1 Ejemplo de una evaluación de una tesis digital

Para poder ejemplificar cada uno de los pasos del proceso de evaluación, presentamos a continuación el proceso y los resultados obtenidos en el análisis de una de las tesis[4] del corpus utilizado.

El primer paso para la evaluación consiste en la creación de la lista de referencia para cada uno de los documentos. En nuestro ejemplo, ésta lista de referencia contiene los conceptos que al experto le parecen como pertinentes. Algunos de los conceptos que definimos como pertinentes para la tesis son los siguientes: *"algoritmos genéticos"*, *"aprendizaje automático"*, *"documento científico"*, *"memoria de instancias"*, *"reformulación de preguntas"*, *"sistema de búsqueda"*, *"sistema multiagente"*, *"técnica de aprendizaje"*, etc. Esta lista de referencia está compuesta por 31 conceptos.

Una vez que tenemos ésta lista, el segundo paso consiste en aplicar cada una de las herramientas y comparar en una tabla si los conceptos que se encuentran dentro de la lista de referencia han sido extraídos por cada una de las herramientas. La tabla correspondiente, Tabla 3, quedaría de la siguiente forma una vez aplicada cada una de las herramientas al documento:

Conceptos de la lista de referencia	XTS	Copernic Summarizer	Terminology Extractor	Nomino
Algoritmos genéticos	X	X	X	X
Aprendizaje automático	X		X	
Memoria de instancias	X	X		X
Reformulación de preguntas	X	X	X	
Sistema de búsqueda	X		X	X
...				

Tabla 3. Ejemplo de comparación de conceptos extraídos por cada una de las herramientas contra los conceptos que se encuentran en la lista de referencia.

El tercer paso consiste en contar el número de conceptos extraídos al igual que las diferencias entre la lista de referencia. Es así como obtenemos para cada uno de los documentos analizados una tabla como la siguiente (Tabla 4):

[4] Tesis correspondiente al ejemplo dado en la sección 5.2: Jeribi L. "Aide à la recherche documentaire adaptée à l'utilisateur". INSA de Lyon, Francia, 7 de Diciembre, 2001.

	XTS	Copernic Summarizer	Terminology Extractor	Nomino
Número total de conceptos extraídos	6932	99	3137	71
Número total de conceptos presentes en la lista de referencia (a)	30	9	19	13
Número total de conceptos ausentes en la lista de referencia (b)	6906	90	3118	58
Número total de conceptos no extraídos (c)	1	22	12	18

Tabla 4. Resultados obtenidos en la evaluación de la tesis.

Por último para poder hacer la comparación entre las herramientas (Sección 5.3) los resultados obtenidos en el tercer paso son aplicados a las métricas de *"precisión"* y *"recuperación"* presentadas en la Sección 4.2. Por ejemplo, para la tesis presentada obtendríamos los siguientes resultados (Tabla 5):

	XTS	Copernic Summarizer	Terminology Extractor	Nomino
Precisión	0.004	0.09	0.006	0.183
Recuperación	0.967	0.29	0.612	0.419

Tabla 5. Ejemplo de porcentajes de *"precisión"* y *"recuperación"* obtenidos en la evaluación de la tesis.

Para este caso, con los resultados obtenidos podríamos decir que la máxima *"precisión"* es la obtenida por Nomino con un 18%, es decir que un 18% de los conceptos extraídos por Nomino son pertinentes. En cambio, XTS tiene un 96% de probabilidades de que entre los conceptos que extrae estén los pertinentes. Como lo muestra la Tabla 4, XTS es la herramienta que más términos extrae ya que selecciona todos los términos que aparecen dentro del documento analizado.

5.3 Comparación de las herramientas

En esta sección presentamos los resultados de la evaluación realizada a las cuatro herramientas seleccionadas. Esta comparación es llevada a cabo utilizando los valores descritos anteriormente en la Sección 4.2, es decir que nos basamos en la *"precisión"* y en la *"recuperación"* para realizar la evaluación.

Una manera para evaluar los resultados, presentados en la Tabla 6, es comparar la eficiencia de cada una de las herramientas contra la eficiencia obtenida al aplicar la herramienta Nomino. Podemos apreciar que la *"precisión"* obtenida por Nomino es mucho más alta que la obtenida por las otras tres herramientas (TerminologyExtractor, XTS, Copernic Summarizer).

	XTS	Copernic Summarizer	Terminology Extractor	Nomino
Precisión	0.028	0.339	0.068	0.834
Recuperación	0.905	0.51	0.648	0.651

Tabla 6. Resultados en términos de *"precisión"* y *"recuperación"* obtenidos al aplicar las cuatro herramientas al corpus de entrada.

Los resultados de la Tabla 6 muestran que la *"precisión"* más alta es la obtenida por Nomino. En comparación con otras herramientas, Nomino extrae menos conceptos irrelevantes. La *"precisión"* más baja es la obtenida por XTS ya que extrae muchos conceptos que son irrelevantes.

En términos de *"recuperación"* XTS obtiene el mejor valor, sin embargo, uno de los inconvenientes es la gran lista que genera ya que cuando se tienen más de 100 conceptos a evaluar el trabajo manual se vuelve largo y tedioso. Por ejemplo, en la Sección 5.2.1 presentamos el caso de una tesis en la que XTS extrajo 6932 conceptos de los cuales casi todos estuvieron presentes en la lista de referencia. Esto se debe en gran medida a que XTS más bien extrae todas las palabras utilizadas en el documento y que no aparecen en la lista de palabras a no extraer. La herramienta TerminologyExtractor tiene el mismo problema que XTS, ambos extraen muchos términos que en la mayoría de los casos no son conceptos. La herramienta Copernic Summarizer es la herramienta que menos conceptos pertinentes extrae ya que sólo selecciona los conceptos que máximo están formados por dos palabras. Siendo bajo el número total de términos que Copernic Summarizer extrae entonces en la gran mayoría de los casos llega a tener un porcentaje de *"recuperación"* que se acerca a más del 50%.

La herramienta Nomino extrae pocos términos al igual que Copernic Summarizer pero al contrario de ésta última herramienta, Nomino extrae conceptos que muchas veces no están en la lista de referencia pero que son pertinentes a utilizar. De esta forma, consideramos que el uso de Nomino podría servir para que inicialmente con los conceptos que extrae se cree una lista de referencia. Además, Nomino es la única herramienta capaz de extraer conceptos que son más complejos. Por ejemplo, en una de nuestras listas iniciales de referencia el experto escogió el siguiente término como pertinente: *"lenguaje de operacionalización Def*"*. En este caso, sólo Nomino pudo extraerlo.

En la Figura 1, presentamos la diferencia entre la *"precisión"* y la *"recuperación"* obtenida por cada herramienta. En este caso, XTS obtuvo el 2.8% de *"precisión"* contra 90.5% de *"recuperación"*. Esto significa que XTS extrae un alto número de términos y que sólo el 2.8% de todos ellos son conceptos pertinentes. Para la herramienta Nomino obtuvimos un 83% de *"precisión"* contra 65.1% de *"recuperación"* lo cual significa que más de la mitad de los términos extraídos por Nomino son conceptos que aparecen en la lista de referencia y por tanto son pertinentes. TerminologyExtractor obtuvo sólo un 6.8% de *"precisión"* y Copernic Summarizer presenta un 33.9% de *"precisión"*. No obstante que Copernic Summarizer tiene muy buena *"precisión"*, aún queda lejos de la que podemos obtener con Nomino que es de un 83.4%.

En conclusión para nosotros la herramienta que extrae más conceptos pertinentes del total de términos extraídos es Nomino. Esta herramienta también presenta la característica de que a través de los términos extraídos genera una red semántica que podría ser de gran utilidad para la generación de índices.

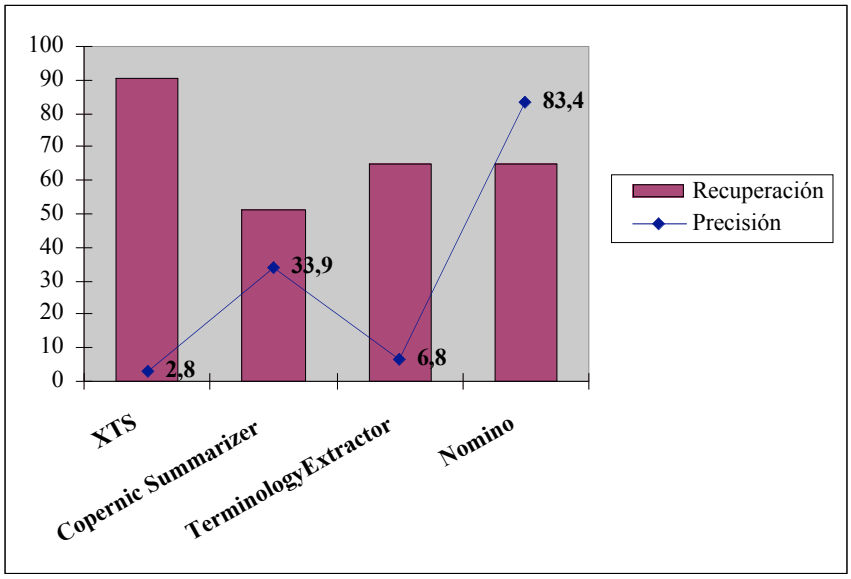


Fig. 1. Comparación de los porcentajes obtenidos en términos de precisión y recuperación.

En la Sección 6 presentamos un prototipo de herramienta de anotación para documentos en formato XML.

6 Herramienta de anotación

El análisis de cada una de las tesis se basa en el estudio de los capítulos o secciones que contienen una mayor cantidad de conceptos interesantes. Al hacer uso de Nomino, nos hemos dado cuenta que las partes correspondientes a la introducción y a la conclusión no son de gran interés ya que son sólo un resumen de toda la tesis. En este caso cuando analizamos la tesis por completo, Nomino no extrae ciertos conceptos ya que se repiten tantas veces en la introducción y en la conclusión que los toma como si fueran conceptos comunes y no pertinentes. Con el objetivo de extraer el máximo número de conceptos pertinentes hemos eliminado la introducción y la conclusión de las tesis. De igual manera eliminamos la página correspondiente a la portada y la bibliografía. En la Tabla 7 se resumen los resultados obtenidos del análisis de la estructura lógica (generalizando a 5 capítulos). Podemos ver que tanto el índice, la introducción y la conclusión no son de gran importancia, al contrario de los primeros dos capítulos que aportan más del 20% de los conceptos que pueden ser utilizados para caracterizar por completo la tesis .

Indice	Introducción	Capítulo 1	Capítulo 2	Capítulo 3	Capítulo 4	Capítulo 5	Conclusión
8.8%	7.8%	20.5%	22.4%	17.2%	18.6%	18.3%	9%

Tabla 7. Comparación de la media de porcentajes obtenidos para cada una de las partes que constituyen la estructura lógica de la tesis.

El análisis de la estructura semántica esta basado en la forma en la que el tesista organiza su información. Hay una cierta tendencia a incluir ciertos elementos siempre en la tesis, más cuando se trata de una tesis del área de informática. Por ejemplo, la gran mayoría de las tesis contienen una sección dedicada al *"modelo"* o al *"método"*. El último capítulo es generalmente consagrado al *"prototipo"* o llamado *"estudio de casos"*. En este último capítulo también encontramos una

sección dedicada a la *"arquitectura"* del prototipo. Encontramos que generalmente el capítulo 2 se enfoca al *"estado del arte"*, el cual presenta el tema principal de la tesis así como la investigación que existe a su alrededor. Como podemos ver en la Tabla 7, éste capítulo es el de mayor porcentaje, es decir el que tiene el mayor número de conceptos interesantes y que por si mismos ofrecen bastante información al usuario. Para el análisis semántico, las tesis fueron divididas de acuerdo a los temas tratados. Por ejemplo, *"estado del arte"*, *"métodos"*, *"arquitectura"*, *"modelos"*, *"prototipo"*, *"metodología"*, *"modelización"*, *"experimentación"*, etc. Estos temas pueden aparecer en diferentes secciones o capítulos. Nomino fue utilizado para extraer los conceptos más importantes.

Con el objetivo de hacer uso de los conceptos extraídos de las tesis a partir del uso de las herramientas evaluadas, hemos propuesto una herramienta capaz de *"anotar"* las tesis con los conceptos pertinentes extraídos por Nomino. El objetivo principal consiste en la utilización de un grupo de tesis en formato XML (eXtensible Markup Language) al cual se agregarán automáticamente los conceptos extraídos en forma de etiquetas XML. De esta manera, cuando el párrafo que contiene el concepto es identificado entonces es anotado con una simple etiqueta al principio como *"<nombre-del-concepto>"* y *"</nombre-del-concepto>"* al final. Este esquema de anotación es muy simple y puede ser aplicado fácilmente a un texto usando un editor de XML, sin embargo nuestra herramienta lo hace automáticamente. Los usuarios pueden validar los conceptos propuestos por Nomino y también pueden proponer nuevos conceptos para ser añadidos en forma de etiquetas. Si éstos nuevos conceptos no aparecen dentro del documento entonces son añadidos como etiquetas XML sólo alrededor del resumen de la tesis con el fin de que una vez efectuada una búsqueda, éste párrafo sirva para darle una idea al usuario de la pertinencia de la tesis.

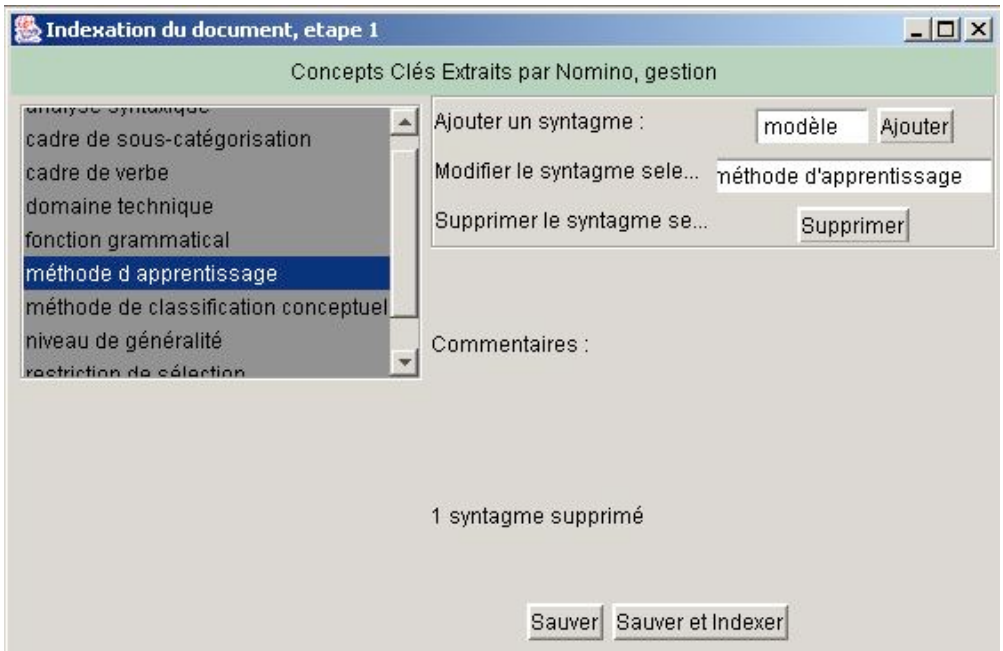


Fig. 2. Indexación y validación de los conceptos a través de la herramienta de anotación.

La herramienta de anotación permite el manejo de los conceptos propuestos por Nomino, la indexación y la extracción de los párrafos pertinentes del documento de acuerdo a un cierto criterio de búsqueda. La Figura 2, presenta la herramienta de anotación propuesta. En esta ventana, el

usuario puede visualizar los conceptos propuestos al igual que puede borrarlos o modificarlos. Cuando se realiza una modificación se requiere de una confirmación. Así, las modificaciones son guardadas y los nuevos conceptos son añadidos a la lista.

Los nuevos conceptos generados por el usuario son guardados e incluidos como etiquetas XML dentro del documento inicial. Esto permite la recuperación de los párrafos que contienen la información pertinente a la hora de efectuar una búsqueda (Figura 3).

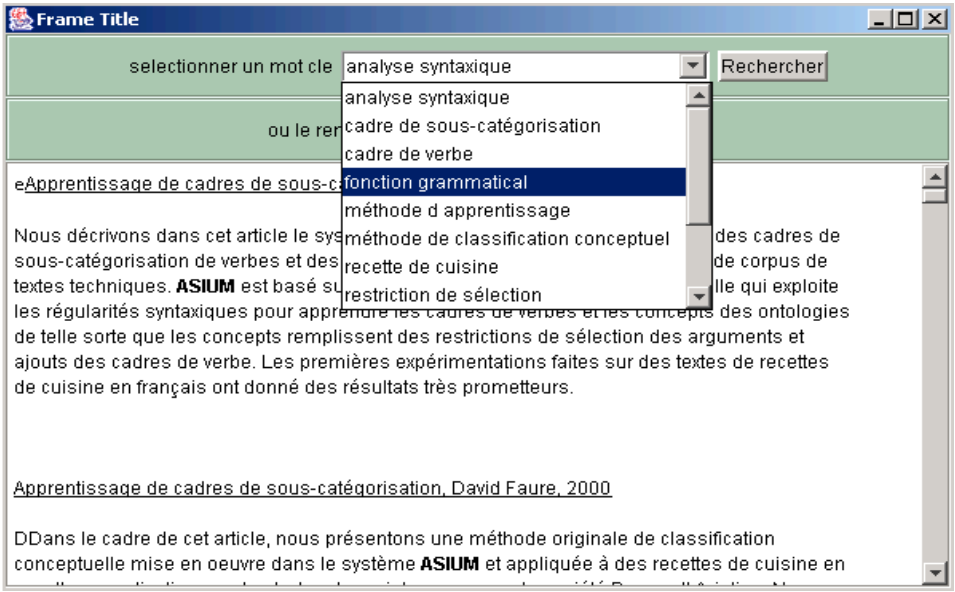


Fig. 3. Recuperación de información basada en los conceptos añadidos como etiquetas XML.

Consideramos de gran importancia el uso de XML Schema para estructurar las tesis de acuerdo a un modelo válido. El propósito del estándar XML Schema es definir la estructura de los documentos XML que estén asignados a tal esquema y los tipos de datos válidos para cada elemento y atributo. De esta forma, hemos construido un modelo usando XML Schema que define una tesis de acuerdo a su estructura lógica (introducción, capítulo, párrafo, frase, etc.) y de acuerdo a su estructura semántica (arquitectura, método, modelo, sistema, etc.). No todas las tesis tienen la misma estructura por lo cual, por ejemplo, hemos definido dentro de XML Schema la etiqueta *"<metadato>"* la cual puede aparecer dentro de cualquier frase no importando su ubicación dentro de la tesis. Así mismo, XML Schema nos permite declarar *"<metadato>"* como una etiqueta que puede aparecer cuantas veces sea necesaria. La etiqueta *"<nombre-del-concepto>"* es considerada como una cadena de caracteres correspondiente a la etiqueta *"<metadato>"*.

En la siguiente sección, Sección 7, presentamos las conclusiones y nuestro trabajo futuro.

7 Conclusión y trabajo futuro

La búsqueda de la información dentro de una biblioteca digital está focalizada al uso de palabras claves que son construidas a medida que el usuario va obteniendo resultados. Generalmente, el usuario no conoce lo que desea buscar y está esperando que se le propongan opciones.

Dentro de la biblioteca digital CITHER, nuestro trabajo consiste en ofrecerle al usuario la posibilidad de interactuar con la colección de tesis a través de temas que nosotros llamamos *"conceptos"* en lugar de la interacción actual que se lleva a cabo a través del título de la tesis, el nombre del autor, la fecha y el año de edición. Para llevar a cabo ésta interacción, necesitamos herramientas capaces de extraer automáticamente los conceptos ya que manualmente hemos comprobado que es un trabajo largo y tedioso. Con el fin de escoger una herramienta adecuada a nuestras necesidades hemos realizado una evaluación de herramientas de extracción automática de términos.

Este artículo presenta la evaluación de cuatro herramientas de extracción automática de términos o conceptos. Las herramientas evaluadas son: (1) TerminologyExtractor de Chamblon Systems Inc., (2) Xerox Terminology Suite de Xerox, (3) Nomino de Nomino Technologies y (4) Copernic Summarizer de NRC. Algunas de las conclusiones a las que llegamos gracias a los resultados obtenidos por medio de esta evaluación son las siguientes:

- XTS extrae una lista muy grande de términos considerados como pertinentes ya que generalmente extrae todos las palabras compuestas que están bien formadas y que para XTS son términos con el simple hecho por ejemplo de ser un sustantivo seguido de un adjetivo. Sin embargo, éstos términos la gran mayoría de las veces no corresponden a conceptos validos.

- Los conceptos extraídos por Nomino están la mayor parte del tiempo formados por más de dos palabras. Estos conceptos son muy interesantes ya que son mucho más específicos.

- Los conceptos extraídos pueden permitir la evaluación y la agregación de nuevos conceptos que faltaban a la lista inicial de referencia.

- Las herramientas evaluadas pueden ayudar al usuario a describir los temas tratados en un documento a partir del uso de conceptos.

- Finalmente, Nomino es la herramienta más adecuada a nuestras necesidades ya que la gran mayoría de los términos que extrae corresponden a conceptos pertinentes. Al generar una lista corta de términos extraídos, Nomino nos permite una evaluación rápida. Además Nomino extrae *"conceptos complejos"* que las otras tres herramientas son incapaces de extraer.

Los resultados obtenidos en esta evaluación no pueden ser generalizados en otras situaciones de trabajo sin antes hacer un análisis adicional. Dentro de nuestro trabajo, hemos sólo evaluado las capacidades para la extracción automática de conceptos aún cuando algunas de las herramientas evaluadas como lo es Copernic Summarizer realizan otro tipo de tratamientos, sin embargo éstos no estaban contemplados dentro de nuestros objetivos iniciales.

Actualmente estamos trabajando en la concepción de un sistema *"inteligente"* basado en los conceptos extraídos que son modelados como etiquetas XML dentro de cada tesis. Así mismo, estamos creando una ontología basada en los conceptos extraídos por Nomino. Las etapas de la concepción de la ontología y algunos primeros resultados son presentados en [1]. Nuestro sistema *"inteligente"* permitirá ofrecerle al usuario los conceptos apropiados, usando la ontología, para efectuar una búsqueda y así obtener la información pertinente.

Dentro del trabajo futuro consideramos también de gran importancia la introducción de un tesauro capaz de aumentar la pertinencia de las palabras usadas al efectuar una búsqueda gracias al uso de conceptos similares o sinónimos.

Referencias

[1] R. Abascal, B. Rumpler, J-M. Pinon. Conception d'une Ontologie dans le Contexte d'une Bibliothèque Numérique. ISKO 2003 (International Society for Knowledge Organization), Grenoble, France, July 3-4, 2003.

[2] R. Abascal, B. Rumpler, J-M. Pinon. Improving information retrieval in digital theses using metadata. International Conference on Electronic Publishing (ELPUB 2002). Karlovy Vary, Czech Republic, Elpub 2002 Proceedings pp. 307-316, ISBN 3-897-0035, November 6-8, 2002.

[3] D. Bourigault, C. Fabre. Approche Linguistique pour l'Analyse Syntaxique de Corpus. Cahiers de grammaire 25, pp. 131-151, 2000.

[4] J. Carlberger et al. Improving Precision in Information Retrieval for Swedish using Stemming. 13th Nordic Conference on Computational Linguistics (NoDaLiDa'01), Upsala, May 21-22, 2001.

[5] Copernic Summarizer 2.0, Copernic Technologies Inc, updated in December, 2001. [online] Available at: <http://www.copernic.com/en/products/summarizer/> (24/08/2004).

[6] B. Daille, J. Royauté, X. Polanco. Evaluation d'une Plate-forme d'Indexation de Termes Complexes. Traitement Automatique des Langues (TAL), 41(2), pp. 395-422, 2000.

[7] E. Frank et al. Domain-Specific Keyphrase Extraction, Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99), Morgan Kaufmann, ed., pp. 668-673, ISBN:1-55860-613-0, 1999.

[8] K. Frantzi, S. Ananladou. Automatic Term Recognition using Contextual Cues. Third DELOS Workshop. Cross-Language Information Retrieval. Zurich, Suisse, March 5-7, 1997.

[9] C. Gutwin et al. Improving browsing in digital libraries with keyphrase indexes. Journal of Decision Support Systems, 27, pp. 81-104, 1999.

[10] S. Jones, G. W. Paynter. Human evaluation of Kea, an automatic keyphrasing system. Proceedings of the First ACM/IEEE-CS Joint Conference on Digital Libraries, Roanoke, Virginia, June 24-29, 2001, ACM Press, pp.148-156.

[11]S. Jones, S. Lundy, G. W. Paynter. Interactive Document Summarisation Using Automatically Extracted Keyphrases. Proceedings of the 35th Hawaii International Conference on System Sciences, 2002.

[12] M. C. L'Homme. Nouvelles technologies et recherche terminologique, Techniques d'extraction des données terminologiques et leur impact sur le travail du terminographe. L'impact des nouvelles technologies sur la gestion terminologique, University York, Toronto, August 2001.

[13] Nomino 4.2.22, updated in July 25, 2001. [online] Available at: <http://www.ling.uqam.ca/nomino/> (24/08/2004).

[14] C. Orasan. Building Annotated Resources for Automatic Text Summarization, Proceedings of the Third International Conference on Language Resources and Evaluation (LREC-2002), Las Palmas de Gran Canaria, España, May 2002.

[15] A. Ribeiro. V. Fresno. A Multi Criteria Function to Concept Extraction in HTML Environment. IC'2001, Las Vegas Nevada, USA. Volume 1, pp. 1-6, 2001.

[16] G. Salton. M. McGill, Introduction to modern information retrieval, McGraw-Hill Book Company, 1983.

[17] TerminologyExtractor 3.0. Chamblon Systems Inc. [online] Available at: <http://www.chamblon.com/terminologyextractor.htm> (24/08/2004).

[18] M. Van Campenhoudt. Les voies de recherche actuelle en terminologie et en terminotique. 7e Université d'Automne en Terminologie, En bons termes, Paris, La Maison du dictionnaire, pp. 109-119, 1998.

[19] Xerox Terminology Suite 2.0. XTS the Terminology Suite, updated in February, 2001. [online] Available at: <http://www.mkms.xerox.com/> (24/08/2004).

Sliding Mode Controller for Robust Force Control of Hydraulic Actuator with Environmental Uncertainties

Jaouad Boumhidi* Mostafa Mrabti†

Abstract

In this paper, a reduced order linear model is selected to describe the hydraulic servo-actuator with large environmental uncertainties. The exploitation in simulation of the perturbed 5^{th} order linear model is enough for the first approach, that is to say, before experimentation to value the studied law control potential. Because its robust character and superior performance in environmental uncertainties, a sliding mode controller, based on the so called equivalent control and robust control components is designed for control of the output force to track asymptotically the desired trajectory with no chattering problems. A comparison with H-infinity controller shows that the proposed sliding mode controller is robustly performant.

Keywords*:* Sliding mode control, hydraulic Servo-Actuator, output tracking.

1 Introduction

Electro hydraulic actuators are widely used in industrial applications [2], [11]. They can generate very high forces and exhibit rapid responses. However, it is well-know that is a complex system with regard to nonlinearity [3]. The linearization based method has been suggested as an effective way of using the nonlinear model of the system in the control law. However, the linearized model is an approximation of the real system dynamics. The latter having uncertainties, the sliding mode controller (SMC) is then preferred because it's robust character and superior performance [5]-[7]. Sliding mode utilizing discontinuous feedback controllers can be used to achieve robust asymptotic output tracking [5], [10]. However, for experimentation, the fast dynamics in the control loop which were neglected in the system model, are often excited by the fast switching of the discontinuous term causing the so called “chattering". The boundary layer solutions are proposed in [6], [9] as chattering suppression method. However the error convergence to zero is not guaranteed. Another class of techniques is based on the use of an observer [4], [5]. However, state observer can cause loss robustness. The higher-order sliding mode approach, known as r-sliding mode is also used [13], [14]. However, the discontinuity set of controllers is a stratified union of manifolds with codimension varying in the range from 1 to the relative degree r. Unfortunately, the complicated structure of the controller discontinuity set causes certain redundant transient chattering. In this study, a new sliding mode controller form is proposed to achieve, both, robust asymptotic output tracking with rapid convergence and with no chattering problems. The control action consists of the equivalent control and robust control components. By an appropriate choose of the later as a continuous function, the chattering problems are eliminated and asymptotic tracking holds guaranteed. By applying the proposed controller, the perturbed sliding surface equation is enforced to zero and by an appropriate choice of this surface, the tracking error tends asymptotically to zero in finite time and with no chattering problems. The organization of this paper is as follows: In section II we present the uncertain

* L.E.S.S.I, Département de PhysiqueFaculté des Sciences Dhar El Mehraz B.P: 1796, 30000 Fes-Atlas, Morocco, jboumhidi@caramail.com

† L.E.S.S.I, Département de PhysiqueFaculté des Sciences Dhar El Mehraz B.P: 1796, 30000 Fes-Atlas, Morocco, mrabti_lessi@yahoo.fr

system model. Section III presents the proposed sliding mode approach, with the design algorithm, Section IV gives the simulation results for the tracking output force by using the SMC which is compared to H-infinity technique [1]-[2], [8] and the concluding remark are given at the end of the paper.

2 System model and preliminaries

The hydraulic system which is the object of this study is composed of a servo valve and actuator, with input voltage and output actuator force. The input voltage modulates the servo valve drawer position, opening supply and return orifices, allowing flow to enter and leave the actuator, which allows the displacement of the piston to create the output force (Fig.1).

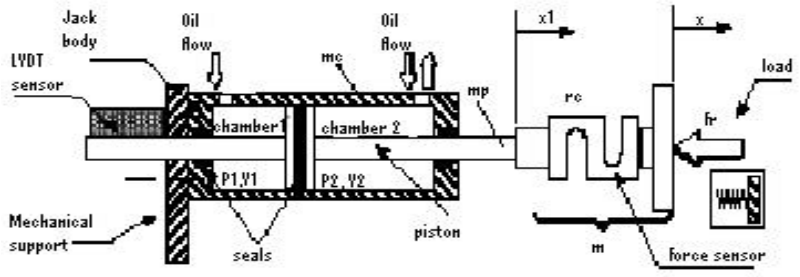


Fig.1 A schematic diagram of the actuator

The system analysis and its nonlinear model are presented in [8]. By linearizing this model equations in the vicinity of an appropriate point of functioning, we obtain the following system equations (1)-(5):

The relation between the servo valve drawer position x_v and the input voltage u can be written as $\frac{x_v}{u} ? \frac{k_v}{s^2 / ?_v^2 ? 2?_v s / ?_v ? 1}$ (1)

where k_v is the valve gain, $?_v$ is the damping ratio of servo valve and $?_v$ is the natural frequency of the servo valve.

The differential equations governing the dynamics of the actuator are:

$$m_p \frac{d^2 x_1}{dt^2} ? SP_u ? f_{eq} \frac{dx_1}{dt} ? r_c (x_1 ? x) \quad (2)$$

where $P_u ? P_1 ? P_2$ is the load pressure, f_{eq} is the spring coefficient and S is the piston ram area.

The relation between the piston and the uncertain environment:

$$m \frac{d^2 x}{dt^2} ? r_c (x_1 ? x) ? r_e x \quad (3)$$

the environmental amortisement is neglected here, r_e is an arbitrary value of the environmental stiffness and $r_c(x_1 ? x)$ is the output force

The relation between the pressure P_u and the flow

$$Q ? S\frac{dx_1}{dt} ? \frac{V_r ? V_m}{2?}\frac{dP_u}{dt} \quad (4)$$

Where V_r the residual volume in the extreme position of the piston, V_m is the mean volume in the mean position of piston and $?$ is the bulk modulus of oil.

The relation between the flow and the servo valve drawer position

$$Q ? K_c x_v ? K_d P_u \quad (5)$$

with $K_c ? \frac{C_q n L_v \sqrt{P_a ? P_{u0}}}{\sqrt{?}}$, $K_d ? ? \frac{1}{2}\frac{C_q n L_v x_{v0}}{\sqrt{?}\sqrt{P_a ? P_{u0}}}$

where K_c and K_d are respectively the flow gain and the pressure coefficient, C_q represents the discharge coefficient, P_a is the supply pressure, n and L_v are the geometric parameters, $?$ is the fluid density and P_{u0}, x_{v0} are respectively the pressure and the valve position of the linearization point.

By combining the equations (1)-(5) and by considering the numeric values of the system parameters [2], we obtain the 7th order linear model defined by the transfer function as:

$$G_7(p) ? \frac{3.817e16p^2 ? 1.614e21}{p^7 ? 2195p^6 ? 5.01e7p^5 ? 9.17e10p^4 ? 2.647e14p^3 ? 4.067e17p^2 ? 1.237e20p}$$

where its frequencies characteristic presents an intrinsic classic aspect of the hydraulic actuator [12].

Many industrial applications, consider for synthesis, the reduced order linear model of hydraulic actuator generally between 2 and 5, in [2], the 3th and 5th order linear models are proposed by considering respectively the 3 first and 5 first poles. The order reduction is operated with regard to there frequencies characteristic and the reduced order linear model are obtained from the empiric approach “Engineering judgment”. According to the frequencies characteristic for these models (Fig. 2), only the 5th order linear model takes into account the localized resonance in approximately 2300 (rad/s). Let consider this reduced 5th order linear model with non minimum phase which is defined by the transfer function as:

$$G_5(p) ? \frac{8.77e8p^2 ? 3.71e13}{p^5 ? 1846p^4 ? 5.944e6p^3 ? 9.324e9p^2 ? 2.843e12p}$$

To obtain a nominal model with minimum phase, the propitious bilinear transformation $p ? \frac{p_t ? k_1}{(p_t / k_2) ? 1}$ with k_1=0.005 and k_2 infinite is considered, which allows the displacement of the poles and zeros in the left half complex plane, without changing their imaginary part. This transformation goes hand in hand with the behaviour of hydraulic actuator in environment uncertainties, because the 7th linear model order has all poles in the left half complex plane. On the other hand, environmental amortisement which allows the zeros in the left half complex plane is neglected in the 7th linear model order.

We obtain the 5th linear model order with minimum-phase where its exploitation in simulation in the presence of uncertainties, is enough for the first approach, that is to say, before experimentation to value the studied law control potential [2].
The nominal model in state space is then as follows:

$$\begin{aligned} \dot{X}(t) &? AX(t) ? Bu(t) \\ y(t) &? KX(t) \end{aligned} \quad (6)$$

where $X ? ?^{n}$ is the available state, with n=5, $u(t) ? ?$, $y(t) ? ?$ and A, B and K are matrices of appropriate dimensions.
The system can be described by the uncertain model as follows:

$$\dot{X}(t) ? AX(t) ? ?AX(t) ? Bu(t) ? ?Bu(t) \quad (7)$$

$$y(t) ? KX(t) ? (?K)X(t) \quad (8)$$

$(?K)X(t)$ is the bounded perturbation term allocating the controlled output force, caused mainly by the large environmental uncertainties.
We denote now the output tracking error by: $e(t) ? y(t) ? y_r(t)$, where $y(t)$ is the controlled output and $y_{r(t)}$ is the reference output. We define the relative degree l of the system to be the least positive integer i for which the derivative $y^{(i)}(t)$ is an explicit function of the input $u(t)$, such that:
$\frac{?y^{(l)}(t)}{?u} ? 0$ *and* $\frac{?y^{(i)}(t)}{?u} ? 0$ *for* $i ? 0,..., l ? 1$. We have:

$$e^{(i)}(t) ? (K ? ?K)(A ? ?A)^{i} X ? y_r^{(i)}(t) \quad \text{for } i ? 1,..., l ? 1$$

With $e^{(0)} ? e(t)$ and $y_r^{(0)}(t) ? y_r(t)$

$$e^{(l)}(t) ? (K ? ?K)(A ? ?A)^{l} X ? y_r^{(l)}(t) ? ?u(t)$$

Where $? ? (K ? ?K)(A ? ?A)^{l?1}(B ? ?B) ? 0$ for all $?A, ?B$ *and* $?K$

Remark. Let: $? (z_0,..., z_{l?1}) ? z_{l?1} ? ?_{l?2} z_{l?2} ? ... ? ?_1 z_1 ? ?_0 z_0$ where the coefficients $?_i$ are chosen so that the characteristic equation $s^{l?1} ? ?_{l?2} s^{l?2} ? ... ? ?_1 s ? ?_0$ has roots strictly in the left half complex plane, then: $? (e, \dot{e},..., e^{l?1}) ? 0$ is the stable linear ordinary equation $e^{(l?1)} ? ?_{l?2} e^{(l?2)} ? ... ? ?_1 \dot{e} ? ?_0 e ? 0$. Then, the output tracking error $e(t)$ tends

asymptotically to zero in a finite time, if we can find a controller which ensures that $\Phi(e, \dot{e}, ..., e^{(l-1)}) = 0$ for *all* $t \geq t_f$ where t_f some finite time $t_f > t_0$

3 Main results

We want that the evolution of the tracking error to be governed by a globally asymptotically stable differential equation, so called sliding surface equation. The main idea is to find a sliding mode controller for the system defined in state space by (7)-(8) which ensures that the sliding surface equation tend asymptotically to zero in a finite time. By an appropriate choice of this surface, the tracking error tends asymptotically to zero in a finite time with no chattering problems. The surface can be expressed as:

$S = \{X(t) : \Phi(e(t), \dot{e}(t), ..., e^{(l-1)}(t)) = 0\}$ Where l is the relative degree of the system, and $\Phi(e, \dot{e}, ..., e^{(l-1)})$ the sliding surface equation which can be selected as follows:

$$\Phi(t) = \Phi(e(t), \dot{e}(t), ..., e^{(l-1)}(t)) = e^{(l-1)}(t) + \lambda_{l-2} e^{(l-2)}(t) + ... + \lambda_1 \dot{e}(t) + \lambda_0 e(t)$$

where the coefficients λ_i are selected according to the above Remark.

$\Phi(t)$ Can be written as: $\Phi(t) = R(\alpha, K)X - Y_r$ (9)

Where: $Y_r = y_r^{(l-1)}(t) - \sum_{i=0}^{l-2} \lambda_i y_r^{(i)}$ and

$$R(\alpha, K) = (K - \hat{K})((A - \hat{A})^{l-1} + \sum_{i=0}^{l-2} \lambda_i (A - \hat{A})^i) \quad (10)$$

$R(\alpha, K)$ Can be written as $R = [r_1\ r_2\ ....r_{n-1}\ r_n] = [r_1 \quad R_2]$ where $R_2 = [r_2 ...r_n]$ and X can be written as $\begin{bmatrix} x_1 \\ X_2 \end{bmatrix}$ where $X_2 = [x_2 ... x_n]^T$, then $\Phi(t)$ can be written as:

$\Phi = r_1(\alpha, K)x_1 + R_2(\alpha, K)X_2 - Y_r$. Let Φ_0 the solution of the equation $\Phi(t) = 0$ with respect to x_1, and then we can specify the sliding surface equation as:

$$\Phi = r_1(x_1 - \Phi_0(X_2, t)) \quad (11)$$

Where $\Phi_0(X_2, t) = (\frac{1}{r_1})(-R_2 X_2 + Y_r)$ with assuming that $r_1 \neq 0$ for all αA and αK.

Using (9) we have $\dot{\Phi}(t) = FX + \dot{Y}_r + \Gamma u(t)$

$$\dot{\Phi}(t) = F_2(\alpha, K)X_2 + \dot{Y}_r + f_1(\alpha, K)x_1 + \Gamma(\alpha, B, K)u(t) \quad (12)$$

Where $\Gamma(\alpha, B, K) = R(\alpha, K)(B - \hat{B})$ and

$$F = R(A - \hat{A}) = [f_1\ f_2 ... f_{n-1}\ f_n] = [f_1 \quad F_2] \quad (13)$$

Theorem: For the system defined by (7)-(8), the sliding mode control law which ensures that $e(t)$ tends asymptotically to zero in finite time can be written as: $u(t) = u_{eq}(t) + u_r(t)$ where:

$u_{eq} ? (? \frac{1}{?})(F_2 X_2 ? f_1 ? _0 ? \bar{Y}_r)$ is the equivalent linear control term which makes the undisturbed nominal system state slide on the S.

$u_r(t) ? ? \frac{m}{?} ? (t)$ Is the term forcing the system to remain on the sliding surface, where the constant m is chosen such that $\frac{f_1}{r_1} ? m$ where r_1 and f_1 are determined respectively from the equations (10) and (13), with $r_1 ? 0$ for all $? A, ? K$.

Proof. In sliding surface, where $? ? 0$, $u ? u_{eq}$ is the control law obtained from the equivalent control method [5] which is determined from the solution of equation $\dot{?}(t) ? 0$ in (12) and assume that $? (t) ? 0$ in (11), we obtain $u_{eq} ? (? \frac{1}{?})(F_2 X_2 ? f_1 ? _0 ? \bar{Y}_r)$.

For the disturbed sliding surface equation $? ? 0$, let us consider a Lyapunov function $V(?) ? \frac{?^2}{2}$. From (12), and by using the expression of $u(t)$ in the theorem we have:

$\dot{V}(?) ? f_1 ? (x_1 ? ? _0) ? m ? ^2$.

Using (11) and choosing m such that $\frac{f_1}{r_1} ? m$ we have $\dot{V}(?) ? 0$.

Design Algorithm:

1. Choose the desired trajectory $y_r(t)$ and formulate the derivative $\bar{y}_r(t), ..., y_r^{(l)}$.

2. Choose the coefficients $? _i$ and formulate the sliding surface equation according to the above Remark

3. From (9), Solve the undisturbed sliding surface equation $? ? 0$ with respect x_1 obtain $? _0$

4. Derive r_1 from (10), F_2 and f_1 from (13) and choose the constant m satisfying the condition in the theorem

5. Formulate the equivalent control and robust control presented in the theorem, respectively for the undisturbed and disturbed sliding surface equation

In conclusion, for the perturbed sliding surface equation, if the constant m satisfies the condition in the theorem, the robust asymptotic convergence is obtained in finite time and the asymptotic tracking will be achieved. Since the proposed robust control term in the theorem is to be used , the chattering will be eliminated and asymptotic tracking will hold guaranteed. In sliding surface, $x_1 ? ? _0$ and the total control tend to the equivalent linear control which makes the undisturbed nominal system state slide on the S.

4 Simulation Results of Hydraulic Servo-Actuator

For the hydraulic servo-actuator described by the uncertain model (7)-(8), the relative degree is $l ? 3$, we have:

$u_{eq}(t) ? (-\frac{1}{?})(f_1 ? _0(t) ? ? \sum_{2}^{5} f_i x_i ? C^T Y_r)$, $u_r(t) ? ? \frac{m}{?} ?(t)$, $m ? 1$ and the sampling step: $? ? 4.10^{?5}$.

$F ? (K ? ?K)((A ? ?A)^3 ? ? _1(A ? ?A)^2 ? ? _0(A ? ?A)) ? ? f_1 \quad f_2 \quad f_3 \quad f_4 \quad f_5 ?$

$C ? ? 1 \quad ?_1 \quad ?_0 ?^T$ with $?_1 ? 1.7$, $?_0 ? 0.7$,

$? ? (K ? ?K)(A ? ?A)^2 (B ? ?B)$, $Y_r(t) ? ? y_r(t) \quad \dot{y}_r(t) \quad y_r(t) ?^T$ and

$R ? (K ? ?K)(A ? ?A)^2 ? ? _1(K ? ?K)(A ? ?A) ? ? _0(K ? ?K)) ? ? r_1 \; r_2 \; r_3 \; r_4 \; r_5 ?$

$$A ? \begin{?} ? 1846.025 & ? 1451.1887 & ? 555.75 & ? 330.97 & ? 1.654 ? \\ 4096 & 0 & 0 & 0 & 0 \\ 0 & 4096 & 0 & 0 & 0 \\ 0 & 0 & 512 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 ? \end{?}$$

$B ? ? 2048 \quad 0 \quad 0 \quad 0 \quad 0 ?^T$,

$K ? ? 0 \quad 0 \quad 1079.75 \quad 0.05e ? 3 \quad 0.05e ? 5 ?$,

$y_r(t)$ is the reference square signal, and $y(t)$ is the controlled force output.

The figures 4 and 6 illustrate the output force when the control laws in figures 3 and 5 are applied respectively. And illustrate the robust asymptotic tracking with no chattering problems for the proposed robust sliding mode controller, which is compared to H-infinity technique presented in [2]. The rapid convergence for the proposed sliding mode controller is also shown.

The simulation results show that the maximal value of the control energy is less than the saturation value of the servo-valve, that is: $u_S ? 3.25$ V [2].

In practice, the perpetual excitations in the control laws in figures 3 and 5, are due to the compensation of the delay registered in the hydraulic-zeros, operated by the injection of an additive tension control.

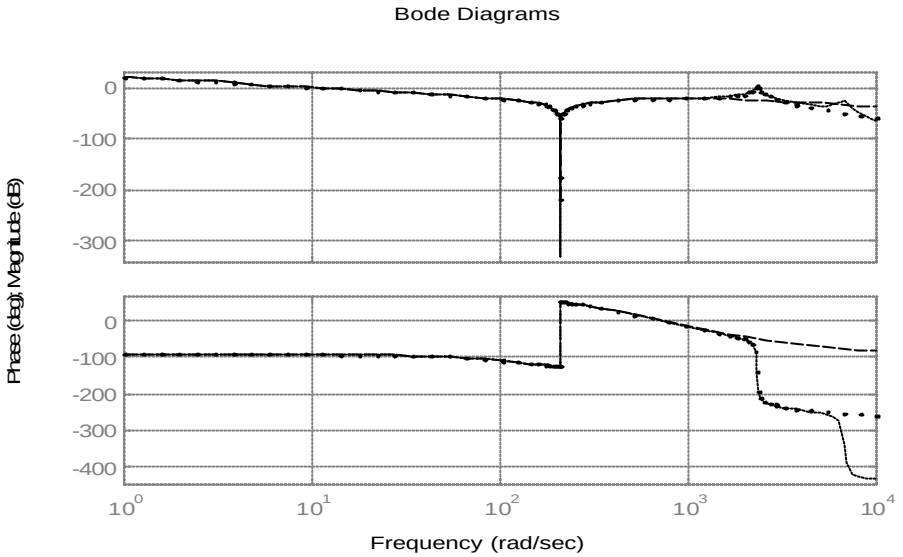


Fig.2. Frequencies characteristic of models (order 7 solid (-)), (order 5 (..)) and (order 3 (--))

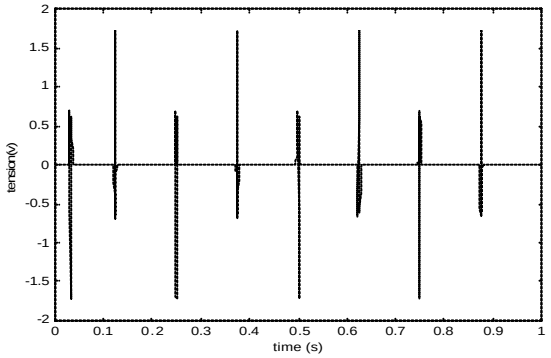


Fig. 3. Sliding mode control law ($?A ? 0, ?B ? 0$ *and* $?K ? 0$)

$$\max(|u(t)|) ? 1.7223(V)$$

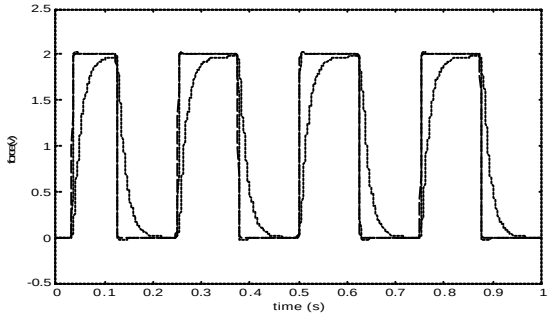


Fig. 4. Output force y(t) (-) and y1(t) (..)when respectively the (SMC) and the H-infinity controller are used

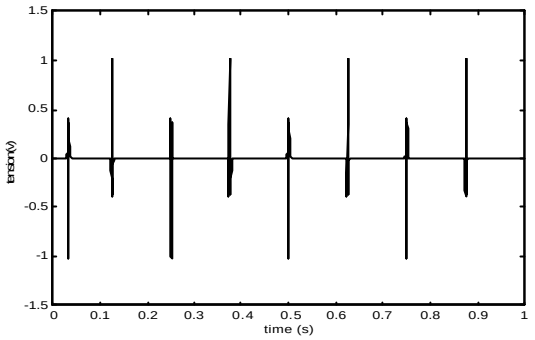


Fig. 5. Sliding mode control law ($?A ? 0.1A, ?B ? 0.1B$ *and* $?K ? 0.3K$)

$$\max(|u(t)|) ? 1.0195\,(V)$$

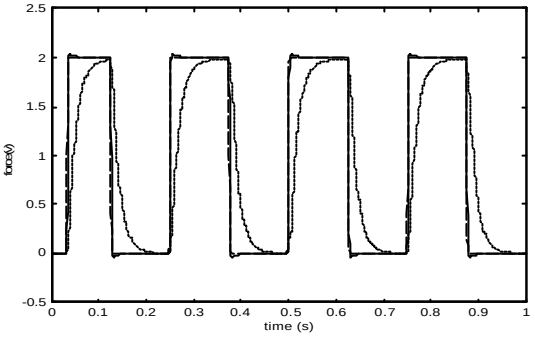


Fig.6. Output force y(t) (-) and y1(t)(..) when respectively the (SMC) and the H-infinity controller are used

5 Conclusion

The proposed sliding mode controller is applied for force control of an hydraulic servo-actuator with environmental uncertainties. The system is described for simulation by the uncertain selected 5^{th} order linear model with minimum-phase, which is enough for the first approach that is to say before the experimentation to value the law control potential. By applying the proposed controller form, both, robust asymptotic output tracking with rapid convergence and with no chattering problems are obtained, and illustrated in the simulation results. The best performance and rapid convergence are also demonstrated for the proposed sliding mode controller when it is compared with H-infinity controller. Consequently, the proposed sliding mode controller has the potential to be implemented for experimentation to obtain a very good performance.

References

[1] D. McFarlane and K. Glover. “A Loop Shaping Design procedure using H? synthesis”, IEEE Transactions on Automatic. Control. 37(6) 759-769, 1992.

[2] L. Laval. “Modélisation et commande en force d’un actionneur hydraulique confronté à un environnement incertain”, Phd Thesis, LRP Paris, December 1997.

[3] T. W. McLain, E. K. Iverson, C. C. Davis, and S. C. Jacobsen. "Developpement, Simulation, and validation of a Highly Nonlinear Hydraulic Servosystem Model", ACC, Pittsburg, Pensylvania, pp. 385-391, June 1989.

[4] G. Bartolini and P. Pydynowski. "An improved chattering free VSC scheme for uncertain dynamical systems", IEEE Transactions on Automatic Control. 41, 1220-1226, 1996.

[5] V. I. Utkin, *Sliding Modes in control optimization*, Springer-Verlag, New York, 1992

[6] J. J. Slotine and S. S. Sastry. "Tracking control of non linear system using sliding surface, with application to robotic manipulators", International Journal of Control, 38, 465?492, 1983.

[7] M. O. Efe, O. Kaynak and X. Yu. "Sliding Mode Control of a Three Degrees of Freedom Anthropoid Robot by Driving the Controller Parameters to an Equivalent Regime", Transactions of the ASME, Journal of Dynamic Systems, Measurement and Control, 122(4) 632-640, 2000.

[8] N. K. M'Sirdi, L. Laval and J. Cadiou. "H? control of hydraulic servo-actuator with environmental uncertainties", proc. IEEE Int. Conf. on Robotics and Automation, Mineapolis, Minesota. 1566?1571, 1996.

[9] J. J. Slotine. "sliding controller design for nonlinear systems", International Journal of control. 40(2) 421-434, 1984.

[10] R. M. Hirschorn. "Singular Sliding-Mode Control", IEEE Transactions on Automatic Control. 46(2) 276-285, 2001.

[11] R. Guenther, M. A. B. Cunha and E. R. De Pieri. "Experimental implementation of the variable structure adaptive cascade control for hydraulic actuators", Power Transmission and Motion Control. Bath, England. 349-361, September 1998.

[12] M. Hamy, Régulation Hydraulique, Techniques de l'Ingénieur, traité Mesures et contrôle, pages R7530?1:26, 1991.

[13] A. Levant. "higher-order sliding modes, differentiation and output-feedback control", International Journal of Control. 76 (9/10) 924-941, 2003.

[14] A. Levant. "Universal SISO Controllers with finite time convergence", IEEE Transactions on Automatic Control. 46(9) 1447-1451, 2001.

Ontology-Based Data Integration Methods: A Framework for Comparison

Agustina Buccella[*], Alejandra Cechich[*] and Nieves R. Brisaboa[†]

Abstract

A data integration system provides a uniform interface to distributed and heterogeneous sources. These sources can be databases as well as unstructured information such as files, HTML pages, etc. One of the most important problems within data integration is the semantic heterogeneity, which analyzes the meaning of terms included in the different information sources. This survey describes seven systems and three proposals for ontology-based data integration. An important feature is that all of them use, in some way, ontologies as the way to solve problems about semantic heterogeneity. In this paper, we show similarities and differences among the systems by providing a framework for comparison and classification.

Keywords: Data Integration, Ontology, Semantic Heterogeneity.

1. Introduction

Data integration is concerned with unifying data that share some common semantics but originate from unrelated sources. Necessarily, when we work on data integration, we must take into account a more important and complex concept called "heterogeneity".

Heterogeneity might be classified into four categories: (1) *structural heterogeneity*, involving different data models; (2) *syntactical heterogeneity*, involving different languages and data representations; (3) *systemic heterogeneity*, involving hardware and operating systems; and (4) *semantics heterogeneity*, involving different concepts and their interpretations. Generally speaking, the semantic heterogeneity deals with three types of concepts [14]: the *semantically equivalent concepts*, the *semantically unrelated concepts*, and the *semantically related concepts*. In the first case – semantically equivalent concepts – a model uses different terms to refer the same concept, e.g. synonymous, or some properties are modelled differently by different systems, for example the concept length may be "meter" in one system and "mile" in one another. In the second case – semantically unrelated concepts – the same term may be used by different systems to denote completely different concepts; and in the last case – semantically related concepts – different classifications may be performed, for example one system classifies "person" as "male" and "female" and other system as "student" and "professor".

There are several methods created to address the problem of dealing with different concepts and interpretations. In general, the approaches might be divided into two different branches: approaches using ontologies and approaches without using ontologies (for example using metadata [10,32]). In this paper we will focus on the use of ontologies because of their advantages when using for data integration. Among them:

(1) the vocabulary provided by the ontology serves as a stable conceptual interface to the databases and is independent of the database schemas,
(2) the language used by the ontology is expressive enough to address the complexity of queries typical of decision-support applications,
(3) knowledge represented by the ontology is sufficiently comprehensive to support translation of all the relevant information sources into its common frame of reference, and
(4) the ontology supports consistent management and recognition of inconsistent data.

The term *ontology* was introduced by Gruber [20] as an "explicit specification of a conceptualization". A *conceptualization*, in this definition, refers to an abstract model of how people commonly think about a real thing in the world; and *explicit specification* means that concepts and relationships of an abstract model receive explicit names and definitions.

[*] Universidad Nacional del Comahue, Departamento de Ciencias de la Computación, Neuquén, Argentina, 8300. Email: abuccel,acechich@uncoma.edu.ar

[†] Universidade de A. Coruña, Departamento de Computación, A. Coruña, España, 15071, Email: brisaboa@udc.es

An ontology gives the name and the description of the domain specific entities by using predicates that represent relationships between these entities. The ontology provides a vocabulary to represent and communicate domain knowledge along with a set of relationships containing the vocabulary's terms at a conceptual level. Therefore, because of its potential to describe the semantic of information sources and to solve the heterogeneity problems, an ontology might be used for data integration tasks.

Some surveys on ontology-based systems for data integration can be found in literature. For example, in [38] authors provide a survey specially focusing on the ontology use, mappings and tools, but without providing a comparison of other elements, such as query resolution or architectural issues. Another different survey can be found in [12], but only languages to represent ontologies are compared in this case.

In this paper, we show the state-of-the-art in ontology-based systems for data integration by describing seven systems – SIMS [1,2,3], OBSERVER [30,31], DOME [13,14], KRAFT [19,33,34], Carnot [28,39], InfoSleuth [8,16,41] and COIN [17,18,35] – and three new proposals. We define a conceptual framework for comparison, which takes into account the most important aspects used to achieve data integration. For example, all of the systems produce ontologies in different ways. Some of them define only one ontology while others define multiple ones. In all cases, the ontology describes the application domain by mapping the information sources to other ontologies. Another important aspect is the ontology language. In general, the systems use languages based on Description Logics, although some Web–based languages have recently emerged. In this paper, we will only indicate the language used to represent ontologies without further explanation. For a description and analysis of ontology languages, we refer the reader to [12].

A framework is proposed for evaluating the systems based on the DESMET [24] approach for Feature Analysis, which will be explained in the next section. The paper is organized as follow: Section 2 describes the conceptual framework and introduces the ontology-based systems in terms of the framework's elements. Section 3 presents the comparison of the systems and the proposals using the DESMET evaluation method. Conclusions are discussed in the final section.

2. A Framework for Comparison of Data Integration Approaches

In order to compare the different approaches, we use the DESMET method [24] due to its widespread use in evaluation techniques. The DESMET method is intended to help an evaluator in a particular organisation or academic institution to plan and execute an evaluation exercise that is unbiased and reliable (e.g. maximises the chance of identifying the best method/tool).

DESMET uses the term Feature Analysis to describe a qualitative evaluation. Feature Analysis is based on the identification of the requirements that users have for a particular task and the mapping of those requirements to features that a method/tool aimed at supporting that task should possess. In the context of software methods/tools, the users might be the software managers, developers or maintainers.
An evaluator then assesses how well the identified features are provided by a number of alternative methods/tools. The main processes involved in carrying out a feature analysis are:

1. Select a set of candidate method/tools to evaluate.
2. Decide the required properties or features of the item being evaluated.
3. Prioritise those properties or features with respect to the requirements of the method/tool users.
4. Decide the level of confidence that is required in the results and therefore select the level of rigour required of the feature analysis.
5. Agree on a scoring/ranking system that can be applied to all the features.
6. Allocate the responsibilities for carrying out the actual feature evaluation.
7. Carry out the evaluation to determine how well the products being evaluated meet the criteria that have been set.
8. Analyse and interpret the results.
9. Present the results to the appropriate decision-makers.

A common framework is necessary to make any sort of comparative evaluation of methods and tools. Feature analysis allows the framework to be expressed in terms of a set of common (mandatory and/or desirable) properties, qualities, attributes, characteristics or features for each type of method or tool. After defining a framework, a method or tool has to be judged as to how much support it actually appears to provide for a feature, i.e., to what degree support is present. Once each method or tool has been "scored" for each feature in the framework using some common judgement scale, the results for the methods or tools have to be compared to decide their relative order of merit.

There are two types of features:

1. Simple features that are either present or absent. These are assessed by a simple YES/NO nominal scale.
2. Compound features where the degree of support offered by the method/tool must be measured or judged on an ordinal scale.

Each simple feature can be accompanied by a degree of *importanc*e assessment. Each compound feature must be accompanied by an assessment of its importance and an appropriate judgment scale associated with the degree of existence or *conformance* to a particular feature or characteristic. The importance of a feature can be assessed by considering whether it is mandatory or only desirable. There may be many gradations of "desirability", but in this paper we only use three of them: *Mandatory* (M), *Highly Desirable* (HD) and *Desirable* (D).

A judgement scale for conformance performs two distinct purposes:

1. It defines what level of support of a particular feature is required.
2. It provides the assessor with a consistent measurement scale against which to "score" the feature of a particular candidate.

The granularity of the scale points depends upon the feature that is being looked at, and the actual requirements. In this paper we will use the generic judgment scale defined in the Table 1 for a particular feature.

Generic Scale Point	Definition of scale point	Scale point mapping
Makes things worse	Cause confusion. The way the feature is implemented makes it difficult to use and/or encouraged incorrect use of the feature.	-1
No support	Fails to recognize it. The feature is not supported nor referred to in the specification manual.	0
Little support	The feature is supported indirectly, for example by the use of other tool features in non-standard combinations.	1
Some support	The feature appears explicitly in the feature list of the tools and specification manual. However, some aspects of feature use are not catered for.	2
Strong support	The feature appears explicitly in the feature list of the tools and specification manual. All aspects of the feature are covered but use of the feature depends on the expertise of the development team.	3
Very strong support	The feature appears explicitly in the feature list of the tools and specification manual. All aspects of the feature are covered and the tools provide tailored dialogue boxes to assist the development team.	4
Full support	The feature appears explicitly in the feature list of the tools and specification manual. All aspects of the feature are covered and the tools provide scenarios to assist the development team such as "wizards".	5

Table 1. Generic judgment scale

Our framework for comparison of data integration approaches (or systems) is described and motivated by the main concepts explained in the introduction. All the systems to be compared will be analyzed from viewpoint of a development team, working in an institution with data integration problems. The framework includes relevant features helping the choice of the appropriate system. As Figure 1 shows, the framework is divided into three main features: *architectur*e, *semantic heterogeneity* and *query resolution*. Each feature contains a set of sub-features or items that further describe the framework. Each of them deals with different aspects of the systems in order to describe them and analyze their advantages and disadvantages. The meaning of each feature is:

(1) Architecture:

- **Information Sources**: This feature refers to the information sources supported by the different systems. A main part of an architecture based on data integration are the information sources involved in the integration, that is, the systems use these sources to retrieve the information and return it to the users. The systems have several architectural components within the architecture in order to access the different information sources. In this paper, the types of information sources are divided into two categories: *the State of the Information Sources (SIS)* and *the Type*

of the Information Systems (TIS). The first category divides the information sources into *dynamic* and *static*. When the information sources are dynamic the systems should create some mechanisms so as to know which information is available at a given moment. The second category indicates the three main types of information systems supported by the systems. They are databases, files and HTML pages.

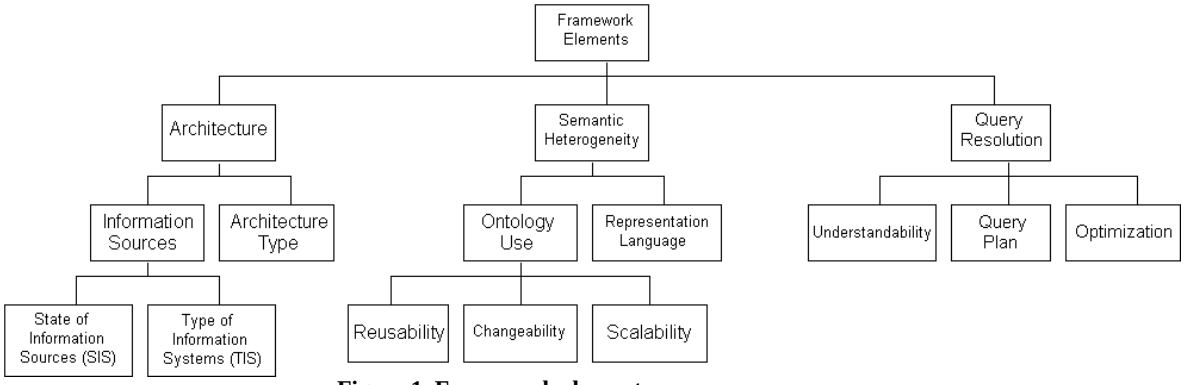


Figure 1. Framework elements

- **Architecture Type**: The software architecture of a system is the structure or structures of the system, which comprise software components, the externally visible properties of those components and the relationships among them [6]. Each system has defined its architectural components based on the user's requirements as well as the need for the working environment. The architecture type can be divided into two types: agent-based and wrapper/mediated-based. Each of them has their own advantages and disadvantages. For example, two main advantages of the agent-based architecture are: (1) agent architectures are designed to allow software processes to communicate knowledge across networks, in high-level communication protocols and (2) agent architectures are highly dynamic and open, allowing agents to locate other agents at run-time, discover the capabilities of other agents, and form cooperative alliances. But this architecture type has also two main disadvantages: (1) the communication among the large number of agents necessary to implement an application may become too expensive and (2) how to understand multiple agents interacting may be difficult, especially if the number of agents is large. On the other hand, the mediator-based architecture has as advantage that all the information needed to achieve the integration is stored in the mediator but this component can make itself appear very complex and difficult to manipulate. Also, performance aspects must be taken into account when mediators are used.

(2) Semantic Heterogeneity:

- **Ontology use**: The first section has described the meaning of the ontologies and how they help solving the data integration problems. The systems should show how the ontologies are used to solve the integration problems and how the ontologies interact with the other components of the architecture. In this feature the system describes its ontological components and the ways to solving the different semantic heterogeneity problems. The development team should find a set of steps or ways to build the ontologies defined in the system. This feature is divided into three sub-features: *reusability*, *changeability* and *scalability*. Reusability refers to the ability of reuse the ontologies, that is, ontologies defined to solve other problems can be used in the system because of either the systems support different ontological languages and/or define local ontologies. Changeability refers to the ability of changing some structures within an information source, without producing substantial changes in the system components. Finally, scalability refers to how easy the integrated system can be extended with new information sources.
  In order to clarify the "Ontology use" feature, Figure 2 shows a diagrammatic representation of part of one ontology used to compare different systems. The arrows in the figure represent properties between two classes. Attributes (or properties relating classes to data types) are represented within class boxes. As we can see, the ontology is modelling an airport's domain.

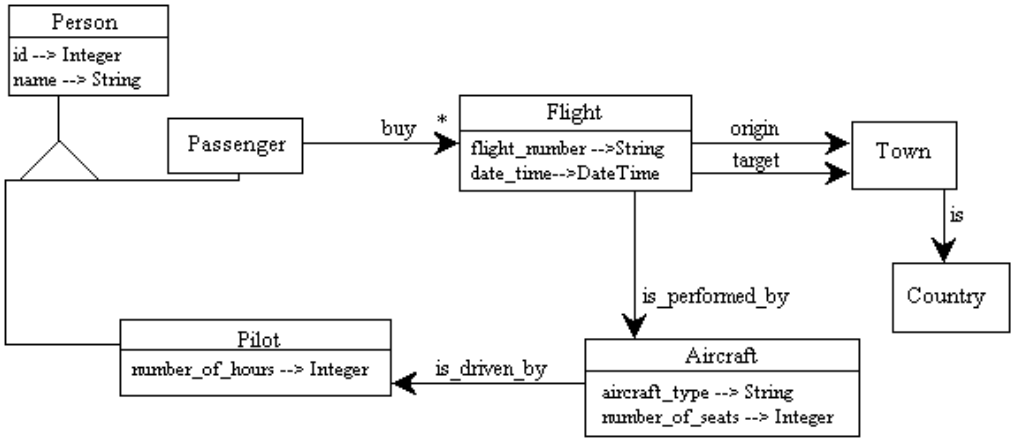


Figure 2. Part of one ontology

Note that Figure 2 only shows a small part of the ontology we have used to illustrate our comparisons.

Now, by applying the sub-features of "Ontology use" to our example, we can see that changeability and scalability impact on some structures by modifying or adding (respectively) the information sources. Depending on the implementation of each system, this change can make the whole ontology be rebuilt or only few modifications be incorporated. Following the example in Figure 2, the next subsections will show how the systems implement the ontologies in order to reach the sub-features.

- **Representation Language**: Each system uses different languages in order to represent the ontologies. Each language has different characteristics with their advantages and disadvantages. The systems should indicate the supported language/s to represent the ontologies and give some support on how the systems use the language/s. For example, either the systems support only one language or support several languages or no restriction of the languages is required. Also, the system should offer some guide to show the language application. Comparison among the languages is out of the scope of this paper. A framework for comparing the expressiveness of the languages can be found in [12].

(3) Query Resolution:

- **Understandability**: In the integrated system users have a unique user interface to access it. Therefore, this user interface should be simple and easy to use. Besides, some guide explaining the interface should be provided. The *understandability* is one of the main quality attribute when user interfaces are thought of. The understandability refers to the capability of the software product to enable the user to understand whether the interface is suitable, and how it can be used for particular tasks and conditions of use. Also, the answer given for the system should be clear and easy for the users, that is, in a language understood by them.
- **Query Plan**: This feature refers to the set of steps needed to achieve a query defined by the users. The system should give a clear description of each step involved showing the interaction among components and ontologies to get a suitable semantic answer. With this feature the development team will understand how the whole system works using the functions of each component.
- **Optimization**: The query optimization can dramatically speed up database query. Many systems have implemented methods to optimize the queries. For example, one method would divide the query into sub-queries, optimize them, and thus generate a faster answer. The comparison among the optimization methods is out of the scope of this paper. We only indicate the optimization methods used by the different systems.

Figure 3 shows the classification into simple and compound features applied to our framework.

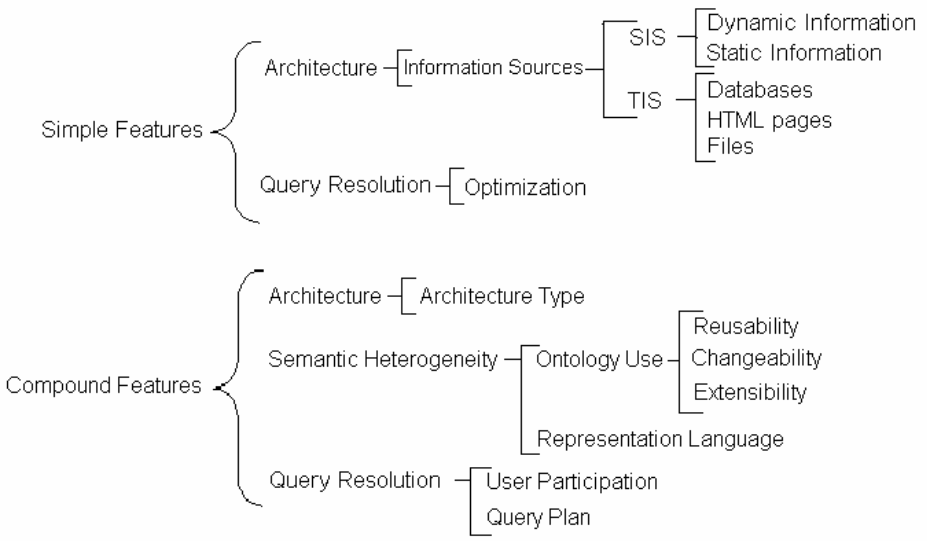


Figure 3. Classification of features between simple and compound

Table 2 shows each feature accompanied by an assessment of its importance. The information sources, architecture type, ontology use and query plan features are mandatory because the minimal needed elements are included within each of these features. When one system does not explain or describe how the ontologies are used, the users or the development team will probably apply them in an incorrect way resulting in an awful system.

Specific Feature	Importance Assessment
Information Sources	M
Architecture Type	M
Ontology Use	M
Representation Language	D
Understandability	HD
Query Plan	M
Optimization	D

Table 2. Importance Assessment for specific features

On the other hand, the representation languages and optimization features are only desirable because the systems may be used properly although these features are incomplete. The same happens with the understandability feature, but in this case the degree of importance is little higher.

In the next sections, we briefly describe seven systems and three proposals, in accordance with the characteristic defined by our framework.

2.1 SIMS (Search In Multiple Sources)

SIMS appeared in 1993 as an information mediator [1,2,3]. The system essentially provides access and integration to multiple sources of information.

Architecture:

- *Information sources*: SIMS was created assuming dynamic information sources, i.e. changing information sources, availability of new information, etc. In general, sources are databases, but SIMS researchers are currently working on including other kinds of information sources, such as HTML pages.
- *Architecture type*: The architecture is based on a *wrapper/mediator*, that is, each information source is accessed through a wrapper. A wrapper, in SIMS, is a module that can translate a data set description into a query, which is submitted to the source. The wrapper also handles communication with the information source. Data returned by the source are taken and sent to the SIMS in some expected format. The SIMS mediator component is used to unify the various available information sources and to provide the terminology for accessing them. The core part of the mediator is the ability to intelligently retrieve and process data.

Semantic Heterogeneity:

- *Ontology use*: A domain model provides the general terminology for a particular application domain. Each information source is incorporated into SIMS by describing the data provided by that source in terms of the domain model. This model is contained in the mediator. SIMS uses a global domain model that also can be called a *global ontology*. The work presented in [38], classifies SIMS as a *single ontology approac*h. In this case, the ontology is composed of the concepts (classes and attributes) that are available from the source, the name of the source that provides the data (also the host, db-name, etc.) and the mappings between them. For example, if two or more sources have attributes with the same values, then they are linked to the same domain concept. A source concept may contain attributes that are not linked to any domain relation.

  Regarding to our example, Figure 2 represents the global ontology for the SIMS system; then any change in the information sources will impact directly on the global ontology. For example, if some information source add information about air-stewardess, the global ontology has to be modified to include this information. Figure 4 shows the changes made to the global ontology in order to represent the new information.

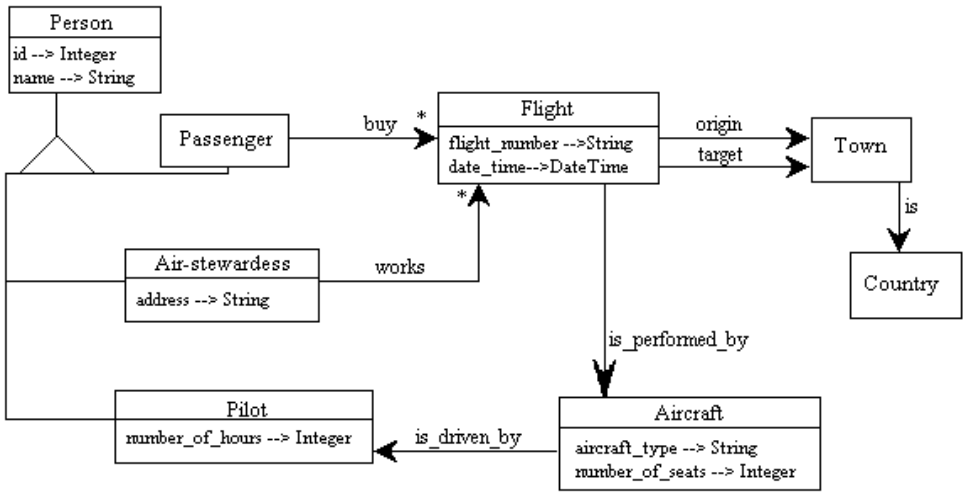


Figure 4. Changes in the global ontology

In this simple example, we add two new elements: the *Air-stewardess* class, as a subclass of the *Person* class, and a property between the *Air-stewardess* class and the *Flight* class. Note that this ontology will be very large, since it will contain all the information in the sources. Therefore, any change or addition on the source's structure will be complex to be implemented in the global ontology.

- *Language*: The domain model (ontology) is described in the Loom language [27]. In Loom, objects in the world are grouped into classes and subclasses. One class can have any number of subclasses, but SIMS does not allow multiple hierarchies. We can define relationships between a subclass and its superclass by explicitly describing the constraints on the subclass. We can also establish relationships between classes. Finally, each class or subclass has a set of attributes.

Query:

- *User participation*: Users make a query in terms of the global ontology without knowing the terms or language used by the underlying information sources. Therefore, queries are written in a high-level language (Loom or a subset of SQL) that actually is the domain model terminology. Queries do not contain information indicating which sources are relevant to their execution or where the sources are located.
- *Query plan*: The first step to answer a query is transforming it into another query expressed in terms of concepts that correspond to information sources. In order to achieve this goal, the system applies four reformulation operations. The *Select-Information-Source* operation maps the concepts in the ontology to the concepts in the information sources, the *Generalize-Concept* operation uses the superclass concept of the ontology when a specific required concept is not in the information sources, the *Specialize-Concept* operation uses the subclass concept in the same way that in the

previous case, and the *Decompose-Relation* operation replaces a particular ontology's relation with the indicated in the information source. The second step consists of generating a query plan and processing the data. To do this, the system divides the query into subqueries trying each of them separately. Once an execution plan has been produced, it is sent to the optimization system.

- *Optimizatio*n: The part of the system that makes the optimization is called *Semantic Query Optimization* (SQO). It optimizes a single subquery and, after that, the entire query plan. SIMS use SQO [21] that can speed up database query answering by using knowledge intensive reformulation.

2.2 OBSERVER (Ontology Based System Enhanced with Relationship for Vocabulary hEterogeneity Resolution)

OBSERVER is an approach that proposes managing multiple information sources through ontologies [30,31].

Architecture:

- *Information sources*: Like SIMS, OBSERVER was created assuming dynamic information sources, but any type of information source such as HTML pages, databases or files, is supported in this case. OBSERVER uses the concept of *data repository,* which might be seen as a set of entity types and attributes. Each repository has a specific data organization and may or may not have a data manager. The different data sources of a repository might be distributed.
- *Architecture typ*e: The architecture is based on *wrappers, ontology servers* and an *IRM* (Inter-ontology Relationship Manager). Here, a wrapper is a module, which understands a specific data organization and knows how to retrieve data from the underlying repository hiding this specific data organization. With Internet sites as data repositories, a wrapper emulates the behaviour of a user, who is accessing data from a Web site. The architecture has several nodes, in which a set of data repositories may have their own wrappers. The Ontology Servers and the IRM will be explained below.

Semantic Heterogeneity:

- *Ontology us*e: In [38], OBSERVER is classified as a *multiple ontology approac*h. OBSERVER defines a model for dealing with multiple ontologies avoiding problems about integrating global ontologies. The different ontologies (user ontologies) can be described using different vocabularies depending on the user's needs. In OBSERVER, every node has only one *Ontology Server*, which is a module that provides information about ontologies located on the node as well as about their underlying data repositories. The system allows encapsulating any direct interaction with user ontologies and also any access to data repositories. Another important component of the OBSERVER system is the *IRM* shared repository. It can be seen as a *catalog of semantics* of the system used to solve the "vocabulary problem" (heterogeneous vocabularies used to describe the same information). OBSERVER also deals with several kinds of interoperable relationships (synonym, hyponym, hypernym, overlap, disjoint and covering).
  In our system, each information source is represented by one ontology; thus a modification or addition of information to some source will only impact on the related ontology and on the IRM. For example, if we add information about air-stewardess, the same changes have to be made to the ontology obtaining the model shown in Figure 4 as a result. But this user-ontology will be smaller than the global ontology in the SIMS system, since it will contain only the information of one source. Therefore, any change or addition on the source's structure will be easier to be implemented.
- *Languag*e: A user may use any language to represent an ontology, but in general, a language based on DL (Description Logics) such as CLASSIC [9] or Loom [27] is used.

Query:

- *User participatio*n: There are two tasks in which a user is involved. During the task *Select User Ontology,* users can browse terms and its definitions, both in DL and in a natural language, in order to choose the terms that exactly fit the semantics of the query. The browse is performed by navigating the different ontologies. During the *Edit Query* task, after selecting the user ontology, the user chooses terms from the ontology to build the constraints and projections that comprise the query. Both tasks together are called *Query Construction.*
- *Query pla*n: The element of the architecture that makes the query resolution is called *Query Processo*r. There are three main steps to answering a query. First of all, the *Query Construction* is performed and it is followed by the *Access to Underlying Data,* and the *Controlled Query Expansion to New Ontologies* steps. The second step translates the user query, expressed in terms of the user ontology, into different queries to the underlying data repositories. To do this, the *Query Processor*

calls the *Ontology Server* that uses some *Mapping Information* to translate the user query, linking the ontology and the underlying data elements. The latest step is executed when the user wants to obtain more relevant data. In this case, the user can choose other ontologies to visit. Then, the original query is translated from terms of the user ontology into terms of another ontology. Obviously the translation is not always exact and the user may define a limit for the *Loss of Information* (loss in precision) allowed.

- *Optimizatio*n: A query is divided into subexpressions that involve only one data repository, although a term in an ontology can be related to several repositories. The cost estimation is concerned with the concept of *Loss of Information* previously mentioned. This concept appears when a user wants to expand the query to other ontologies obtaining more relevant data. Two types of plans can be built in this case: *Plans without loss of information* (when a complete translation of the user query can be achieved, for example by using synonymous) and *Plans with loss of information* (when a complete translation cannot be achieved, for example by using hyponym and hypernym). The user can indicate a desired limit of loss of information (a percentage). The system provides a theoretical basis for estimating information loss measures.

2.3 DOME (Domain Ontology Management Environment)

DOME [13,14] is still under development. It is focused on ontology development by using software reverse engineering techniques.

Architecture:

- *Information sources*: DOME only deals with structured databases and their application programs where the data sources are legacy information systems. However, DOME researchers are currently working on including semi-structured data sources such as Web pages as well.
- *Architecture typ*e: The most important architectural components are wrappers, a set of tools for extracting and defining ontologies and mappings between them (grouped into the Engineering Client component), the Mapping Server, and the Ontology Server. Here, a wrapper performs a translation of queries expressed in the DOME query syntax to queries expressed in the syntax of the data sources.

Semantic Heterogeneity:

- *Ontology us*e: The better approach to describe the DOME system is the *multiple ontology approach* [38]. But regarding to the *Client Engineering* approach, it is also possible to combine a *top-down* and a *bottom-up* ontology development approaches. In this case, the top-down approach identifies key concepts by consulting corporate data standards, information models or generic ontologies such as Cyc or Word Net. This process results in *shared ontologies*, which are used as the common terminology between a number of different systems. The bottom-up approach starts with the underlying data sources and uses ontology extraction tools, such as XRA [42], which are applied to database schemas and application programs to produce initial ontologies. They are further refined into two separate ontologies: *resource ontologies*, which define the terminology used by specific information resources; and *application ontologies*, which define the terminology of a user-group or client-application. Once the ontologies have been defined, they are stored in the Ontology Server. The rest of the ontology engineering task consists of defining a mapping between the resource and the shared ontologies by a particular application. As we previously mentioned, DOME uses XRA [42] as a tool to generate ontologies. XRA is an ontology extraction tool that uses a software reverse engineering approach to extract an initial ontology from given data sources and their application programs.

  In DOME system, like in OBSERVER, Figure 2 only represents an ontology (resource ontology) with semantic information of one source. The modification or addition of a source will change this ontology, and also the mappings between this resource ontology and application and shared ontologies. For example, Figure 5 shows part of one shared ontology. As we can see, it contains information about air-stewardess but it is represented using a different concept (synonym) and a relationship. Therefore to finish the process of adding information, the following mappings have to be added:

  Flight_Assistant ⇔ Air-stewardess
  Flight_Assistant.assists ⇔ Air-stewardess.works

  Some other mappings already stored in the system are:

Airplane_Pilot ⇔ Pilot
Airplane ⇔ Aircraft
Airplane.airplane_type ⇔ Aircraft.aircraft_type
Person ⇔ Person
……..

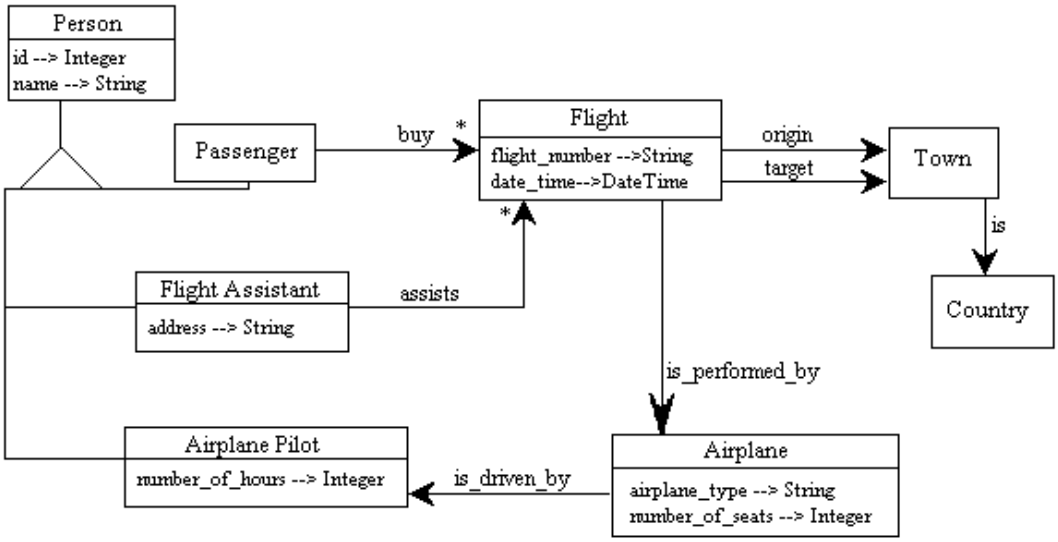


Figure 5. The shared ontology in DOME

- *Language*: The three ontologies previously mentioned are specified using CLASSIC [9], which is used to store ontologies and to make some inferences.

Query:

- *User participation*: DOME has a simple API that allows a user client or an application to query the distributed information space. The user can load and browse specific ontologies to view what information is available from the whole information space. Queries are performed using the terminology defined by an *application ontology* and the returned results are represented using that terminology. However, the details of the different systems – their distribution, structure, syntax or semantics – are actually hidden.
- *Query plan*: The element of the architecture that performs queries is called *Query Engine*. Firstly, the Query Engine needs to decide which resources are relevant to a given query. Then, based on the information about the currently relevant resources, it decomposes the query into subqueries. Results are integrated after been received.
- *Optimization*: There is no cost estimation that shows the best way to achieve a query. As we previously mentioned, the query is divided into subqueries to access the information sources.

2.4 KRAFT (Knowledge Reuse And Fusion/Transformation)

KRAFT [19,33,34] was primarily conceived to support configuration design of applications among multiple organizations with heterogeneous knowledge and data models. It uses the concept of *"Knowledge fusion"* to denote the combination of knowledge from different sources in a dynamic way.

Architecture:

- *Information sources*: Like the other systems, KRAFT was created assuming dynamic information sources. Here, information sources are called *resources* and include both databases and knowledge bases. Resources have constraints that may be stored as metadata of the database, as triggers, etc.
- *Architecture type*: KRAFT has an agent-based architecture, in which all knowledge processing components are realized as software agents. There are four kinds of agents: *wrappers*, *mediators*, *facilitators* and *user agents*. The *wrapper agents* provide a bridge among the legacy system interface, the KRAFT agent interface, and entry-points for user agents. The *user agents* allow end-users to access the information by using some kind of user interface. The *facilitator agents* provide internal routing services for messages within the KRAFT domain. When an agent needs a service, it

asks a facilitator to recommend another agent that provides the service. Finally, the *mediator agents* use the domain knowledge to transform data in order to increase its information content. Examples of data transformation might be integration of data from heterogeneous sources, consistency checking and refinement of knowledge and data, etc.

Semantic Heterogeneity:

- *Ontology use*: The better approach to describe the KRAFT system is the *hybrid ontology approach* [38]. In order to overcome the problems of semantics heterogeneity, KRAFT defines two kinds of ontologies: a *local ontology* and a *shared ontology*. For each knowledge source there is a local ontology. The shared ontology formally defines the terminology of the domain problem. In order to avoid the problems about semantics heterogeneity, that might occur between a local ontology and the shared ontology (*ontology mismatches* [37]), an *ontology mapping* is defined for each knowledge source. It is a partial function that maps terms and expressions defined by a local ontology to terms and expressions of the shared ontology.
  Something similar occurs in the KRAFT system compared with DOME and OBSERVER. The main difference here is that KRAFT contains a shared ontology. When a modification or addition is made in some source, the local ontology (which represents this source) has to be changed, and the mappings between the local and the shared ontology have to be performed. To illustrate this situation we can apply the same example used in the DOME system, because Figure 5 represents one shared ontology in that system. In KRAFT, Figure 5 represents the only shared ontology containing all the information that the system needs to be accessible for users.
- *Language*: The local and shared ontologies may not be represented in the same format, even though they use classical frame-based representation languages.

Query:

- *User participation*: Users access the services of the KRAFT domain via *user agents*. Users specify their requirements as constrains written in any language (KRAFT Constraint Interchange Format, CIF, CoLan languages [7]). So, in order to translate these constraints, the terminology used to perform the queries must be defined in the shared ontologies.
- *Query plan*: The resource constraints must be translated into CIF constraints before the KRAFT network can use it. Communication between agents is made through messages, which include the CIF constraints as part of them. The main function of the facilitator agents is to solve a query called *facilitation*. The goal of this query is to find a resource whose advertised capability matches the requirements derived from the query. The satisfiability criterion depends on the search strategy adopted for the resolution of the query. There are two resolution mechanisms – *the most exact match* and *the set of all approximate matches*.
- *Optimization*: There is no mechanism to provide optimization.

2.5 Carnot and InfoSleuth

The Carnot Project [28,39] was initiated in 1990 with the goal of addressing the problem of logically unifying physically distributed, heterogeneous information. The InfoSleuth project [8,16,41] is an extension of Carnot to make legacy database systems easily accessible via Web.

Architecture:

- *Information sources*: InfoSleuth was created to add the concepts of dynamic information sources to Carnot. It was designed to operate on a static environment, where the information sources never change.
- *Architecture type*: Like KRAFT, these systems have an agent-based architecture to provide interoperation among autonomous systems. There are five kinds of agents in InfoSleuth. The *user agents* assist the user in formulating queries over an ontology, and in displaying the results of queries in a sensitive manner to the user context. The *broker agents* determine the set of relevant resources that can perform the requested service. The *resource agents* translate queries expressed in a common query language into a language understood by the underlying system. These agents usually perform a mapping between the ontology and the local data. The *task execution agents* are designed to deal with dynamic, incomplete and uncertain knowledge. These agents use information provided by a *broker agent* to route requests to the specific *resource agents*. Finally, the fifth type – *ontology agents* – will be explained in the following point.

Semantic Heterogeneity:

- *Ontology use*: Ontologies in InfoSleuth system are used to capture database schemas, conceptual models and aspects of its agent architecture. Here, there are two main tasks to accomplish. The first

one is concerned with *describing the information sources* as the system may have multiple ontologies describing data. The second task aims at *specifying the agent infrastructure*, i.e. the context in which agents operate, its relevant information and relationships, etc. InfoSleuth implements a model with three layers to represent ontologies: the *Frame Layer*, which allows to create and query new meta-models, the *Meta-Model Layer*, in which objects are instances of frame layer objects, and the *Ontology Layer*, in which objects are instances of meta-model objects. Therefore, the most appropriate approach to describe the InfoSleuth system is the *multiple ontology approach* [38]. Otherwise, Carnot is implemented using the *single ontology approach* due to the system provides a global view of the all integrated resources.

Regarding to our example, the same case applied in the SIMS system can be applied in the Carnot system because it also define a global ontology (or view) about the whole domain. In the case of InfoSleuth, the three-layer model is represented by Figure 6, in which the Airport-Ontology box in the Ontology layer is the ontology described in Figure 2.

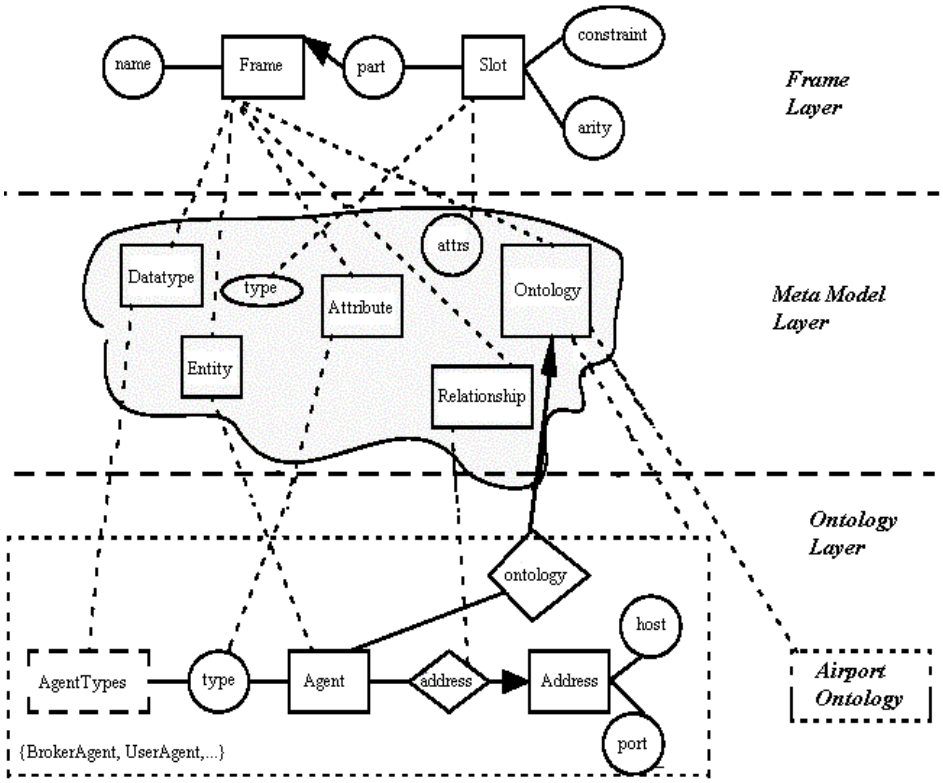


Figure 6. The three-layer model in InfoSleuth

Any change in the subjacent source generates the modification of the Airport Ontology. This case is similar to the application of OBSERVER because each source will have its own ontology, and therefore changes will be easier to be implemented.

- *Language*: There is no specific language required for the system. The ontologies might be represented, for example, by using logic programming languages such as LDL [43].

Query:

- *User participation*: Users interact with the system through a *user agent* specifying requests and queries over given ontologies. In this way, a *user agent* obtains information from an *ontology agent* about the common ontological models known by the system. A user agent uses this information to support its associated user in selecting an ontology and formulating a set of queries.
- *Query plan*: When a user performs a query, the *user agent* asks to the *broker agent* for the location of an applicable *execution agent*. When the *execution agent* receives the request, it asks to the *broker agent* for the location of the *ontology agent* and for an appropriate ontology. Then, the *broker agent* is asked for the currently *resource agent*. Depending on the resource availability, the *broker agent* may return a different set of *resource agents* at different time. Then, based on this set, the *execution*

agent decomposes the query and routes it. Each resource agent, after translating the query, returns the results to the *execution agen*t. After receiving the results, the execution agent integrates and returns these results to the *user agen*t, which in turn returns them to the user. On the other hand, Carnot system uses its *query graph generator module* to generate a query graph. This graph is expanded by a *semantics module* including sources with relevant information. Finally, an optimal plan is generated and executed.

- *Optimizatio*n: The query processing strategy in Carnot is based on a dataflow execution model similar to the strategy used in the ORION distributed object-oriented database system [40].

2.6 COIN (Context Interchange)

The COIN Project [17,18,35] was initiated in 1991 with the goal of achieving semantics interoperability among heterogeneous information sources.

Architecture:

- *Information sources*: The COIN system provides access to both traditional data sources such as databases and semi-structured information sources such as Web sites.
- *Architecture typ*e: The system has a mediator-based architecture. The main elements of this architecture are *wrappers, context axioms, elevation axioms, a domain model, context mediators, an optimizer* and *a executione*r. Like other systems, here a *wrapper* is a module that understands a specific data organization and knows how to retrieve data from the underlying repository hiding that data organization. The other architectural elements will be explained below.

Semantic Heterogeneity:

- *Ontology us*e: COIN introduces a new definition for describing things in the world. It states that *the truth of a statement can only be understood with reference to a given contex*t. Here, the notion of *context* is useful to model statements in conflicting heterogeneous databases. Using this definition, COIN creates a framework that constitutes a formal and logical specification of the COIN system components. The framework has three components, which have been previously introduced in the architectural analysis: a *domain mode*l, *elevation axioms* and *context axiom*s. Each information source has a set of elevation axioms and a set of context axioms, and both converge in a unique domain model. Therefore, the most appropriate approach to describe the COIN system is the *hybrid ontology approach* [38]. The *domain model* is a collection of source's primitive types, such as strings or integers, and semantics types that define the application domain corresponding to the integrated data sources. Primitive objects and semantics objects are instances of primitive types and semantics types respectively. Semantics objects may have properties called *modifiers* that serve as annotations to make the data's semantics of different contexts explicit. Also, semantics objects may have different values in different contexts. The *elevation axioms* act as a mapping between attributes in the source and semantics types in the *domain mode*l. Also, they codify the integrity constraints of the sources, which are useful for producing optimal queries. The *context axioms* define alternative interpretations of the semantics objects in different contexts. Every source is associated with exactly one context, but several sources may share the same context. These axioms are divided into two different groups: (1) axioms that define the semantics of data at the source in terms of values assigned to *modifiers*, which correspond to semantics objects, and (2) axioms that define how values of a given semantics object are transformed between different contexts *(conversion function*s).
  As we have described, COIN has a different representation of an ontology but it is based on an hybrid ontology approach. Therefore COIN can be compared with KRAFT or InfoSleuth, in which a change in one source will generate changes in some components to represent the new mappings. In COIN, elevation and context axioms have to be defined to access the new information.
- *Languag*e: The COIN framework is specified as a deductive and object-oriented data model of the family of F-logics [23].

Query:

- *User participatio*n: Users formulate a query based on an ontology without specifying a priori what information sources are relevant.
- *Query pla*n: The query plan involves the last three elements of the architecture: a *context mediator*, which rewrites the query as a *mediated query;* the *optimizer*, which transforms the mediated query as an optimized plan; and the *executioner* that executes it. The *context mediator* is in charge of the identification and resolution of potential semantics conflicts induced by a query. The automatic conflict detection and reconciliation are both possible by using general knowledge of the underlying application domain, along with the informational content and implicit assumptions associated with

the sources. These bodies of declarative knowledge are represented as elements of the framework *(domain mode*l, a set of *elevation axioms* and a set of *context axiom*s). The result of the mediation is a mediated query. To retrieve the data from the different information sources, the *optimizer* transforms the mediated query into a query execution plan, which is optimized taking into account the topology of the network of sources and their capabilities. Then the *executioner* executes the plan to retrieve the data from the various sources. Results are composed as a message and sent to the user. The mediation is performed by an abductive procedure, which produces a reformulation of the initial query in terms of the component sources. The procedure itself is inspired by the abductive logic programming framework [22].

- *Optimizatio*n: The application of abductive reasoning to query rewriting is a useful tool to achieve efficiency. It can be used to formally combine and to implement features of semantics query optimization (SQO) [11].

2.7. Proposal 1 by Arruda, Baptista and Lima [4]

This proposal presents an architecture and design of a web-based query system in which users, using an ontology, can specify their queries and submit them to the underlying data sources.

Architecture:

- *Information sources*: The approach proposes a mediator-based environment for data integration of structured and semi-structured data on the web. The heterogeneous data sources can be either structured data such relational or object databases, or semistructured data such HTML pages, text documents and XML [5]. The semistructured data are treated as XML documents and therefore it is required that the semistructured data repositories export their data into XML documents.
- *Architecture typ*e: The architecture is a mediator-based data integration system, which uses an ontology as its common schema. The main components of the architecture are: the *search engine*, the *mediator and query rewriting* and the *wrappers.* The first component is responsible for receiving the query from the user; for identifying the user chosen ontology; for iterating with the mediator and query rewriting component to structure and enrich semantically the query and for distributing the query among the wrappers (as in structured data) and calling appropriate XML query engines (as in semistructured data). The second component, the mediator and query rewriting, is responsible for maintaining the integrated schema view (the ontologies); for structuring and rewriting the queries according to the ontology and integrating all distributed results in XML format for a posterior document transformation using a publisher framework. Finally the *wrappers* are specific for each underlying structured source and translate the generic application queries into source specific queries. Also, they translate the results from these queries into XML format and send them back to the mediator module.

Semantic Heterogeneity:

- *Ontology us*e: This proposal uses XML documents as information sources. XML provides only a syntactical view of the underlying data. In order to allow for a precise querying and semantic interpretation, the XML documents should be complemented with a conceptual model that adequately describes the semantics of its tags. The proposal uses the Ontologies to fulfil this semantic gap enabling a precise semantic expression for querying XML documents. The ontologies are used as a common schema.

  In this proposal applies the same example as in the SIMS system, because it also define a global ontology (or global schema) containing information about the integrated sources.
- *Languag*e: The common schema is described in DAML+OIL [15] language and supports several ontologies, related or not. The DAML+OIL language is a semantic markup language for Web resources and is being developed as an extension to XML and the RDF [26].

Query:

- *User participatio*n: Users can access the system through a Web device (PDAs, mobile phones, browsers, etc), they browse it graphically and compose their queries using the specified ontology. The interface is developed for web browsers. A tree is automatically built from the DAML ontology elements. Also, the fields for the input query are built according to the ontology attributes for each element.
- *Query pla*n: The proposal has six steps in order to solve a query:

  (1) The users can access the system through a Web device browsing it graphically and compose their queries using the specified ontology.

(2) The mediator and query rewriting component receives the query and parses it in accordance to the conceptual ontological terms. The query then is rewritten using additional information of the ontology, keeping the semantics of the original query.
(3) The search engine then maps this query (which is generic a priori) into queries for both types of source: queries to semistructured data (XML documents) and to structured data (traditional databases). The query is transferred to the XML query engine, in the case of semistructured data, and in the case of structured data, the query is transferred to the wrapper of each underlying source.
(4) The XML Query Engine accesses all documents structured according to the user chosen ontology and related ones.
(5) Each wrapper maps the generic query into the appropriate query language of the local repository. Hence, it makes the due mapping between the internal data source schema and the external schema (based on the ontology). This mapping is done according to a previous conceptual correspondence between ontology and scheme. The results of these queries are then modelled into XML documents and delivered to the mediator and query rewriting component.
(6) The mediator module receives the results from the two kinds of sources in XML format, integrates them and calls an XML transformer (XSLT, for example) to the specific web device.

- *Optimization*: There is no mechanism to provide optimization.

2.8. Proposal 2 by Medcraft, Schiel and Baptista [29]

This proposal presents an architecture for semantic integration using mobile agents and a global ontology. The global ontology is used to represent the semantic information of the sources, and the mobile agents are used to process queries.

Architecture:

- *Information sources*: The objective of this proposal is to provide a uniform interface to heterogeneous, pre-existing relational databases, in a way that users can formulate queries (global queries) that are transported between these different databases by a mobile agent to produce a single consolidated response.
- *Architecture type*: The proposal has an agent-based architecture, known as DIA (Data Integration using Agents), for semantic integration of a federated database using mobile agents and ontologies. The architecture has the user interface developed with web browsers and the ontology includes the equivalent or similar concepts in the source databases. Also, the architecture proposes the use of a travelling pattern and a task pattern [25]. Travelling patterns deal with various aspects of managing the movement of mobile agents. These patterns allow us to enforce encapsulation of mobility management, which enhances reuse and simplifies mobile agent design. An extended *Itinerary* travelling pattern is implemented because it maintains a list of destinations, always knows where to go next and defines special cases such as what to do if a destination is temporally unavailable. To enable a host to be part of the mobile agent destination list, every destination needs to have an agency running a local stationary agent, which cooperates with the mobile agent in the local query adaptation process. Task patterns are concerned with the breakdown of tasks and how these tasks are delegated to one or more agents. The proposal also uses the *Master-Slave* pattern, which allows a master agent to delegate a task to a slave agent. The slave agent moves along its itinerary, performs the specified task and returns the result to its master agent.

Semantic Heterogeneity:

- *Ontology use*: The proposal identifies the objects in the databases that represent equivalent or similar concepts, that is, concepts semantically related. Heterogeneities between the databases appear as semantic conflicts. They are detected and solved at the moment a new database enters the federation. Once identified the equivalent or similar concepts in the new database, they are defined as global concepts in the existing ontology. This ontology describes the concepts as classes and properties, and defines the relationships between each other.

  In our example, Figure 2 represents the global ontology. When one change in the source is made, this proposal finds the equivalent concept of the global ontology, and modifies the matching table to store this new relationship accordingly. The global ontology proposed here represents the information about the domain. Each information source through the matching table maps its information with the information represented in the global ontology. Therefore, if added information (in this case about air-stewardess) is represented in the global ontology, only the matching table has to be modified.

- *Language*: The ontology is represented by the DAML+OIL [15] language.

Query:

- *User participation*: Users submit queries expressed over the ontology. The interface was developed for web browsers. A tree is automatically built from the DAML+OIL ontology classes and the fields for the input query are also built according to the ontology properties for each class.
- *Query plan*: Users trigger the query process by preparing a global query, based on terms from an ontology. When the user submits the query, the application notifies a stationary agent running inside an agency. This stationary agent (master agent) creates a mobile agent (slave agent) for the migration through the databases. Applying the Itinerary pattern explained in the architecture feature, before its creation, a list of destinations is set for the new mobile agent. With the query and knowing its itinerary, the migration process starts. As the mobile agent reaches a destination, it contacts the local stationary agent, passing it the global query. This agent knows the ontology on which the global query terms are based on, and also knows the local database schema. Therefore, with a local matching mechanism, it is able to convert the global query to a corresponding local one. The database query is performed and the result set presented to the visiting mobile agent is showed in XML syntax. In case the mobile agent is not at the first destination, before moving to the next node, the mobile agent performs an integration of the local result with the result collected previously. This reduces the size of the carried XML, and means that when the mobile agent returns to its origin, no extra result integration or analysis will be performed. When the mobile agent completes its itinerary, it migrates back to its origin, returns the final result to its master agent and removes itself. The master agent presents the collected data to the user.
- *Optimization*: The proposal realizes an optimized itinerary using two types of itineraries: *static* and *dynamic*. A static itinerary is an itinerary that can be established in advance by the stationary master agent. Depending on information available about the local databases, the master agent analyses the global query and selects those local databases that must be visited to process the query. In the simplest case, the itinerary is "visit all nodes". In a dynamic itinerary, the mobile agent can decide, depending on the partial result, if the query is already answered and it can return, or not. This dynamic capability depends on a previous classification of the global query.

2.9. Proposal 3 by Tzitzikas, Spyratos and Constantopoulos [36]

This proposal presents an approach for providing uniform access to multiple information sources through mediators and ontologies. It proposes a new component, named articulation, to improve the cost of adding or modifying the sources.

Architecture:

- *Information sources*: This approach proposes a model for providing integrated and unified access to multiple information sources of the type of Web catalogs. In the environment of the Web there are many examples of such sources. Specifically, the general purpose catalogs of the Web, such as Yahoo! or Open Directory (http://dmoz.org), the domain specific catalogs/gateways (e.g. for medicine, physics, tourism), as well as the personal bookmarks of the Web browsers can be considered as examples of such sources.
- *Architecture type*: The system has a mediator-based architecture. For each source are defined *ontologies* plus a *database* storing objects that are of interest to its users. Each object in the database of a source is indexed under one or more terms of the ontology of that source. Another architecture component is the *mediator*. The mediator is a secondary source that can bridge the heterogeneities that may exist between two or more sources and provides a unified access to those sources. Besides, the mediator has a number of *articulations* to the sources. An articulation to a source is a set of relationships between the terms of the mediator and the terms of that source. These relationships are defined by the designer of the mediator at design time and are stored at the mediator.

Semantic Heterogeneity:

- *Ontology use*: As we have said above each source has an *ontology*, that is, a structured set of names, or *terms*, that are familiar to the users of the source. In particular the ontologies that consider in this proposal consist of a set of terms structured by a subsumption relation. An ontology is a pair $(T, \preceq)$ where T is a *terminology*, i.e. a set of names, or *terms*, and $\preceq$ is a *subsumption* relation over T, i.e. a reflexive and transitive relation over T. In addition to its ontology, each source has a stored *interpretation I* of its terminology, i.e. a function $I : T \rightarrow 2^{Obj}$ that associates each term of T with a

set of objects. The symbol 2^{Obj} is to denote the power set of *Obj*. Also, the mediator has an ontology $(T, \preceq)$ that reflects the needs of its potential users but does not maintain a database of objects. Instead, the mediator has a number of *articulations* to the sources. The relationships included in the articulations are defined by the designer of the mediator at design time and are stored at the mediator. To solve heterogeneity problems among the sources, this proposal does not merge the ontologies, that is, if the same term appears within two sources, they do not assume as equivalent because they may have different interpretations (meanings). Two terms are considered equivalent only if they can be shown to be equivalent using the articulations of each source.

As the most appropriate approach to describe this proposal is the *hybrid ontology approach* [38], the modification or addition of a change in the sources only generates the modification of the local ontology (which represents this source) and the addition of the mappings between the ontology of the mediator and the local ontology within the respective articulation. If we consider that the ontology of the mediator is shown in Figure 5, then the mappings added in this example are the mappings that we have to add in the articulation.

- *Language*: To represent the ontologies there is no language proposed by this approach.

Query:

- *User participation*: Users submit queries expressed over the ontology of the mediator. Upon receiving a user query, the mediator has to query the underlying sources. A user who looks for objects of interest can browse the ontology of the source until he reaches the desired terms, or he can query the source by submitting a boolean expression of terms.
- *Query plan*: Users formulate queries over the ontology of the mediator and it is the task of the mediator to choose the sources to be queried, and the query to be sent to each source. Then it is again the mediator that appropriately combines the results returned by the sources in order to produce the final answer. Specifically, the mediator uses the articulations in order to translate queries over its own ontology to queries over the ontologies of the articulated sources. The sources can provide two types of answers to a given query, namely, a *sure* answer or a *possible* answer (the first type of answer being appropriate for users that focus on precision, while the second for users that focus on recall). Moreover a user query to the mediator admits two types of translation, namely, *lower* or *upper* translation (again, the first type of translation being appropriate for users that focus on precision, while the second for users that focus on recall). The combination of these two types of answer and of translation generates four interpretations: lower-sure, lower-possible, upper-sure and upper-possible. Therefore, the mediator can answer queries submitted by its users based on any of these four interpretations. Also, the operation modes of the mediator either can be decided (and fixed) by the mediator designer at design time, or they may be indicated by the mediator users at query time.
- *Optimization*: There is no mechanism to provide optimization.

3. Discussion

Table 3, and Figure 7 and 8 present the results of the systems evaluation for each feature, i.e. architecture, semantic heterogeneity and query resolution; using the judgment scale explained in Section 2.

Table 3 shows the resultant assessment of the Architecture feature. It has simple features assessed by a simple YES/NO scale. As we can see in the first column, SIS (dynamic information), some systems are assessed by a NO value, that is, these systems do not have implemented mechanisms to know which information is available at a given moment. The second column, TIS, assesses the types of information systems supported by the systems. All systems, except the Proposal 3, support the integration of databases in the federation. But only some of them support semistructured data such as HTML pages and files. The last column, Architecture Type, does not have an assessment because more information is necessary in order to evaluate the architecture properties. We divide the architecture into two main approaches: agent-based approaches and mediator-based approaches and classify the systems into these approaches. For example, the KRAFT system uses an agent-based architecture to provide extensibility and adaptability in a dynamic distributed environment. KRAFT also focuses on the interchange of data and constraints among agents in the system. Another example is the Proposal 2 working on a mobile agent architecture to improve the process of retrieving and consolidating data from the heterogeneous databases. Otherwise, a system like COIN uses a mediator-based architecture to detect semantic conflicts among heterogeneous systems by comparing context axioms corresponding to the sources involved.

	Architecture				
	Information Sources				
	SIS	TIS			
	Dynamic Information	Databases	HTML pages	Files	Architecture Type
SIMS	Yes	Yes	No	No	mediator-based
OBSERVER	Yes	Yes	Yes	Yes	wrappers, ontology server and IRM
DOME	No	Yes	No	No	wrappers, ontology server and mapping server
KRAFT	Yes	Yes	No	No	agent-based
Carnot	No	Yes	No	No	agent-based
InfoSleuth	Yes	Yes	Yes	Yes	agent-based
COIN	Yes	Yes	Yes	Yes	mediator-based
Proposal 1	No	Yes	Yes	Yes	mediator-based
Proposal 2	Yes	Yes	No	No	agent-based
Proposal 3	No	No	Yes	No	mediator-based

Table 3. The results for Architecture feature

Figure 7 shows the results of assessing the ten systems for the semantic heterogeneity feature by using a graphical representation. All of the sub-features are compound features and they are assessed by the judgment scale for conformance defined in Table 2.

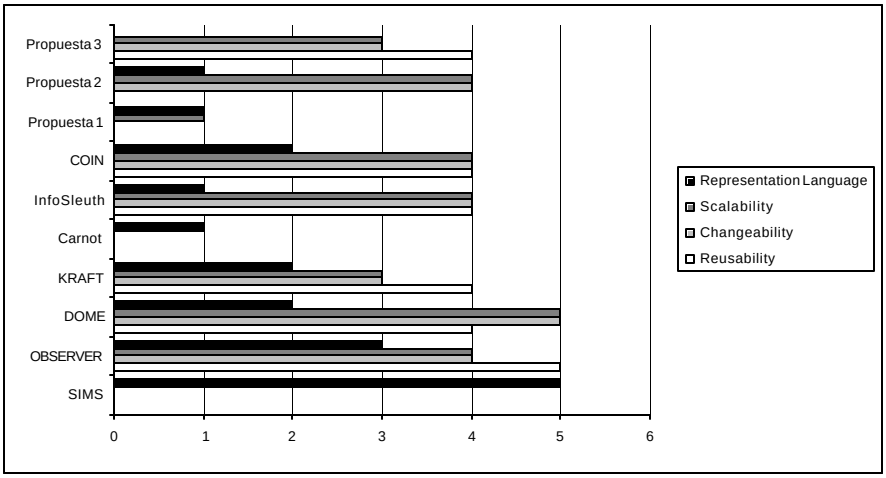


Figure 7. A graphical representation for Semantic heterogeneity feature

We will first analyze the ontology use feature (reusability, changeability, scalability). As we can see SIMS and Carnot systems have a zero (0) in the three sub-features. In the first sub-feature the zero means that both ontologies are not reusable because both define a global ontology or a single ontology approach in order to integrate the data. The same happens with the changeability sub-feature, because no support is given by these systems to bear changes in the information sources. The global ontology must be rebuilt when one source changes. Finally, scalability, is not supported because again by adding a new information source the rebuilding of the global ontology is generated. The problem of dealing with the global ontology approach is that we must manage a global integrated ontology, which involves administration, maintenance, consistency and efficiency problems that are very hard to solve. For example, SIMS requires constructing a general domain model that encompasses the relevant parts of the database schemas. Due to each database model is related to this general domain model, the integration problem is shifted from how to build a single integrated model to how to map the domain and the information source models. Also, the Proposal 1 has a similar assessment because a global ontology is built and changes in one source or the addition of a new source generates a modification in this global ontology.

Otherwise, the OBSERVER system, for example, uses multiple ontologies (with different vocabularies) to solve the users needs in a better way, and to alleviate the problems presented by the single approach. In order

to do so, OBSERVER implements an IRM component that enhances the scalability of the query processing strategy by avoiding the need of designing a common global ontology. The reusability is also full supported by OBSERVER because it defines local ontologies that might be defined for other purposes.

KRAFT has the first sub-feature, reusability, scored with a very strong support (4) because it defines an hybrid ontology approach where the local ontologies can be defined independently. The changeability is assessed as a strong support because when a source changes, only the local ontologies and the mapping ontology must be modified by including the changed information. The scalability is also strongly supported because we only include the information of the new source in the same components. Something similar happens with Proposal 3, COIN and InfoSleuth because in the case of Proposal 3 to add a source we have to select a mediator in the network and design an articulation between the selected mediator and the new source. In the case of COIN the domain model does not have to be updated each time a new source is added. Finally, in the InfoSleuth system to add a resource to the system the resource only needs to have an interface to advertise its services and let other agents make use of it immediately.

The two last sub-features in the DOME system have the higher value, i.e. full support(5) because DOME researchers are developing tools in order to extract ontologies from legacy systems, check the ontology consistency, merge ontologies, etc. All these tools generate a highly scalable system.

Otherwise, Proposal 2 has no support for reusability because a global ontology is defined. But the changeability and scalability have very high values because this global ontology does not need to be modified when a source is added, only an ontology-schema matching table is necessary to add a new database to the integrated system.

The support given by the systems to apply some specified language is assessed by the representation language sub-feature. Only SIMS has a full support because it is the only one that has a full description about the language used in the ontology (Loom). Otherwise, OBSERVER does not have a predefined language, that is, the different ontologies (user ontologies) can be described by using different vocabularies depending on the users needs although it shows several examples and uses of CLASSIC language describing how the system work. The other systems have little or no support for the languages. Some of them only refer to the used language but no help or explanation is provided.

Figure 8 shows the results of assessing the ten systems for the query resolution feature by using a graphical representation. All of the sub-features are compound features and they are assessed by the judgment scale for conformance defined in Table 2.

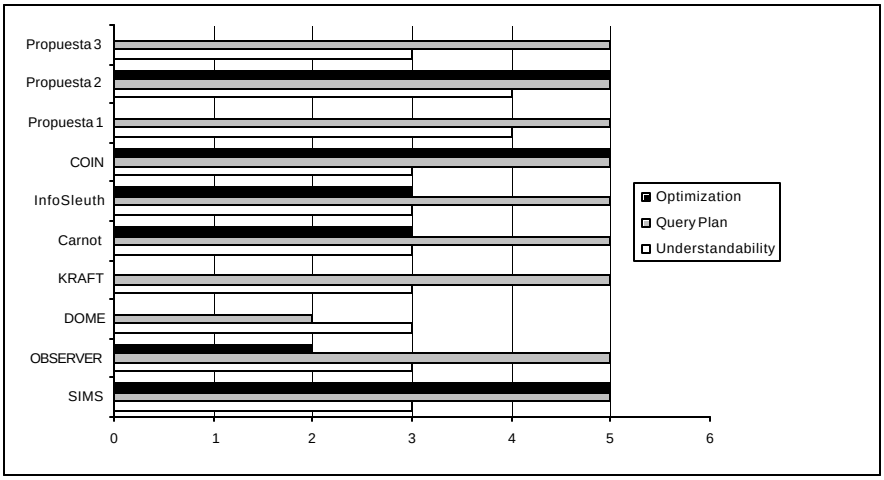


Figure 8. A graphical representation for Query resolution feature

The user participation sub-feature evaluates the degree of understandability that the system provides. In all the systems, users perform a query based on ontologies because of their better understandability. Therefore, as we can see, the first seven systems and the last one have a strong support on this feature. But Proposal 1 and Proposal 2 have a very strong support because the user interface used to make queries is shown and described more clearly. Also, a clear and understandable answer is given by all of them.

The second sub-feature, query plan, evaluates whether the systems clearly describe a set of steps to explain the query process. In general, the query is divided into subqueries in such a way that each of them corresponds to a different information source.

All systems, except DOME have a full description about the query plan describing each step and how the components interact in order to get a consistent answer. The DOME system does not describe the steps to achieve the query, only explain the components of the architecture. It does not show how the components interact in a query resolution.

Regarding to optimization, the last sub-feature, some systems (DOME, KRAFT, Proposal 3 and Proposal 1) do not have any mechanism in order to optimize the query. In the case of OBSERVER, it does not have any mechanism to optimize the query, but uses the concept *loss of information* to describe the situation in which a user wants to expand the query to other ontologies obtaining more relevant data. Indirectly, this characteristic would help the optimization of the query. Other systems like InfoSleuth and Carnot have a similar optimization strategy that ORION system applies. The full support values are chosen for the SIMS, COIN and Proposal 2 systems because all of them have implemented specific mechanisms to optimize the query. For example, SIMS and COIN use SQO to achieve it.

Finally, we should analyze the score sheets and determine which of the systems is the best. Our analysis is based on accumulating the absolute scores. As we have simple features as well as compound ones we need assign a score to the YES and a score to the NO. It is inappropriate to score 1 for providing a simple feature because that will bias your assessment towards systems that provide some support for compound features. Therefore, we assess the YES simple feature with a score 5, and the NO simple feature with a score 0. Besides, we use the following weights to assess the relative importance of features:

- Mandatory: 10
- Highly Desirable: 6
- Desirable: 3

Each feature will be assessed by the product between the assessed value on each feature and the specified weight depending on its importance. For example, in the case of SIMS for *Information Sources* feature, the resultant value is 100 ((5+5)x10, because it is a mandatory feature) out of a possible 200 giving a percentage score of 50%. Another example can be COIN, with a resultant value of 200 (100%) obtained by the product between 20 (5x4, because the four subfeatures are scored with 5) and 10.

Figure 9 graphically shows the percentages for each system within the Information Sources feature. The highest values (100%) are obtained by COIN, InfoSleuth and OBSERVER systems because they fulfil all the subfeatures within the Information Sources feature. The lowest values are achieved by Carnot, Proposal 3 and DOME systems.

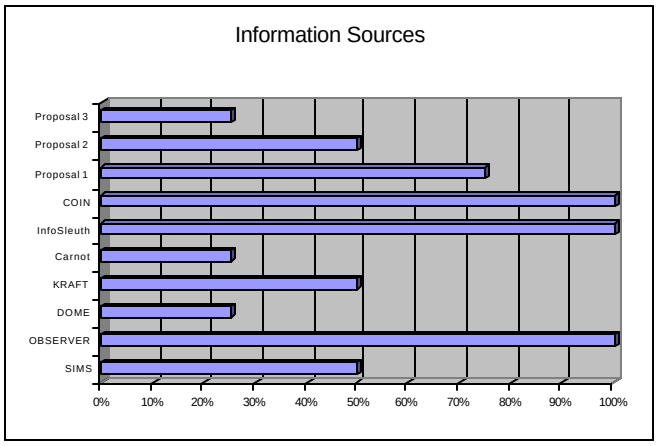


Figure 9. The information sources percentages

Similarly, Figure 10 presents the percentages of the Semantic Heterogeneity feature.

DOME and OBSERVER systems have the highest values while Carnot, Proposal 1 and SIMS systems obtain the lowest values.

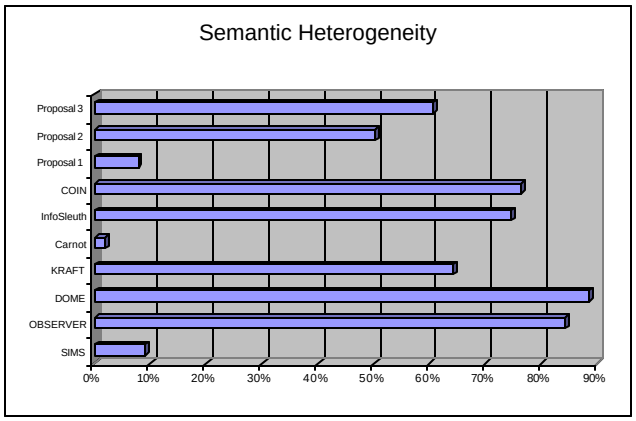


Figure 10. The semantic heterogeneity percentages

Finally, Figure 11 shows the percentages of the Query Resolution feature. All systems have high values but Proposal 2 and SIMS systems are the best.

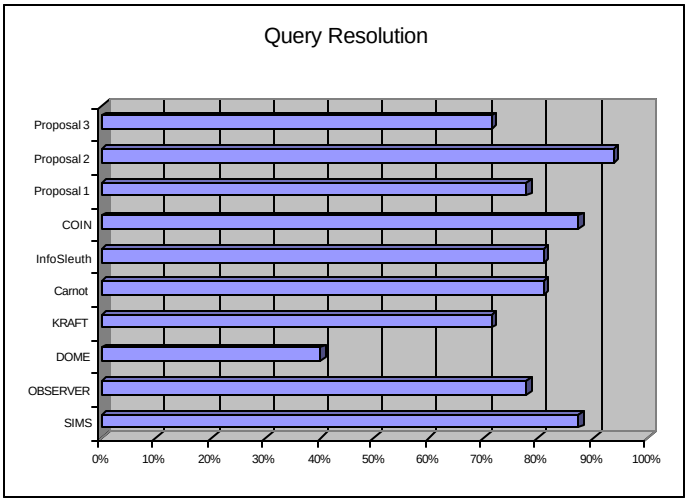


Figure 11. The query resolution percentages

With all of these percentages in mind, we can do the final comparison. Figure 12 presents the total values obtained by the product between the addition of all the assessed features and their respective weights (accordingly with their importance). The higher value allowed joining all features, being 460, that is the 100%.

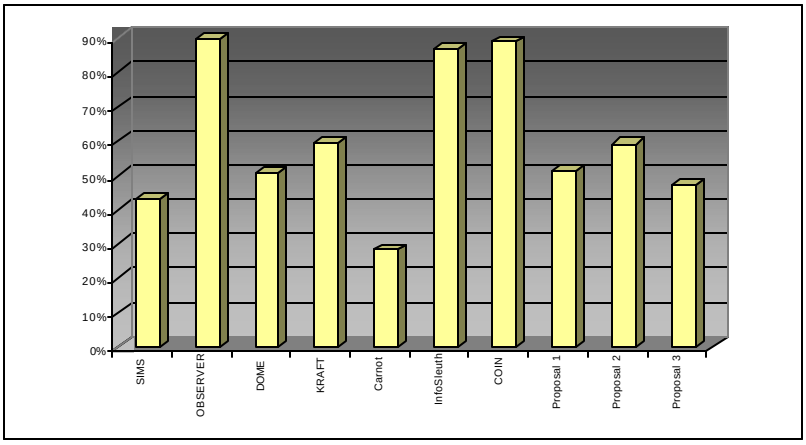


Figure 12. The final percentages

OBSERVER, InfoSleuth and COIN systems have obtained the highest values, between 80-90%, because they have achieved higher scores in many features or subfeatures of the framework. The second range of scores, between 50-60%, has been obtained by Proposal 2, Proposal 1, KRAFT and DOME systems. Finally, Proposal 3, SIMS and Carnot systems have the lower values because some features or subfeatures are not supported.

4. Conclusion

In this paper, we have presented both an analysis and a comparison of seven systems and three proposals that use ontologies to solve the problems involved in data integration. In order to do so, we have created a conceptual framework with three main categories: *architecture*, *semantics heterogeneity* and *query resolution*. Based upon our comparison, we have found some elements in common and also original aspects of the systems.

In order to assess the features defined in our framework the DESMET approach for Feature Analysis has been used. This approach assesses the features with a scale of values according to the mechanism used by the systems to resolve them. OBSERVER, InfoSleuth and COIN systems have obtained the higher values because they have totally achieved many features of the framework.

Our survey intents help for the ontology-based data integration community giving and comparing different aspects of systems which have been used as a reference for further research. But other several aspects have to be analyzed as the valuation of the architecture properties. Also for each approach, a more exhaustive analysis is needed to compare the optimization techniques applied to the query plans.

Some of these systems are still under development, and currently the research community is looking for ways of improving the construction of their ontologies. Therefore, there is still an ongoing concern on how ontology-based data integration should be. We hope our comparison be useful to the increasingly community working today on data integration using ontologies.

References

1. Ambite, J.L., Arens, Y., Ashish, N., Knoblock, C. A. and collaborators. The SIMS Manual 2.0. Technical Report, *University of Southern, California.* December 22, 1997. http://www.isi.edu/sims/papers/sims-manual.ps.
2. Arens, Y., Hsu, C., Knoblock, C. A. Query processing in the SIMS Information Mediator. *Advanced Planning Technology, Austin Tate (Ed.), AAAI Press*, Menlo Park, CA, 61-69, 1996.
3. Arens, Y., Hsu, C., Knoblock, C. A. Retrieving and integrating data from multiple information sources. *International Journal in Intelligent and Cooperative Information Systems*,Vol.2(2), 127-158, 1993.
4. Arruda, L., Baptista, C., Lima, C. MEDIWEB: A Mediator-based environment for data integration on the web. *Databases and Information Systems Integration. ICEIS* , 34-41, 2002.
5. Baru C. Features and Requirements for an XML View Definition Language: Lessons from XML Information Mediation. In: *W3CWorkshop on Query Language(QL´98)*, Boston, 1998.
6. Bass, L., Clements, P., Kazmar, R. *Software Architecture in Practice.* Addison-Wesley. 1998.
7. Bassiliades, N. and Gray, P. M. D. CoLan: A functional constraint language and its implementation, Data and Knowledge Engineering, 14, 203–249, 1994.
8. Bayardo Jr., R. J., Bohrer, W., Brice, R., Cichocki, A., Fowler, J., Helal, A., Kashyap, V., Ksiezyk, T., Martin, G., Nodine, M., Rashid, M., Rusinkiewicz, M., Shea, R., Unnikrishnan, C., Unruh, A. and Woelk, D. InfoSleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments. Microelectronics and Computer Technology Corporation (MCC).
9. Borgida, A., Brachman, R.J., McGuinness, D.L., and Resnick, L.A. CLASSIC: A structural data model for objects. In Proceedings *ACM SIGMOD-89*, Portland, Oregon, 1989.
10. Busse, S., Kutsche, R. D., Leser, U. & Weber H. (1999). Federated Information Systems: Concepts, Terminology and Architectures. Technical Report. Nr. 99-9, *TU Berlin*.
11. Chakravarthy, U. S., Grant, J., and Minker, J. Logic-based approach to semantic query optimization. *ACM Trans. Database Syst.* Vol.15(2), 162–207, 1990.
12. Corcho, O. and Gomez-Perez, A. Evaluating knowledge representation and reasoning capabilities of ontology specification languages. *In Proceedings of the ECAI 2000 Workshop on Applications of Ontologies and Problem-Solving Methods*, Berlin, 2000.
13. Cui, Z., Jones, D. and O'Brien, P. Issues in Ontology-based Information Integration. *IJCAI 2001*– Seattle, USA • August 5 2001.
14. Cui, Z. and O'Brien, P. Domain Ontology Management Environment. *In Proceedings of the 33rd Hawaii International Conference on System Sciences*. 2000.

15. DAML+OIL 2001: Available at:http://www.daml.org/2001/03/daml+oil-index.
16. Deschaine, L., M., Brice, R., S., Nodine, M., H. Use of InfoSleuth to Coordinate Information Acquisition, Tracking and Analysis in Complex Applications. Technical Report *MCC-INSL-008-00*. 2000.
17. Firat, A., Madnick, S., Grosof, B. Knowledge integration to overcome ontological heterogeneity: challenges from financial information systems. *Twenty-Third International Conference on Information Systems*. 2002.
18. Goh, C.H., Bressan, S., Siegel, M. and Madnick, S. E. Context Interchange: New Features and Formalisms for the Intelligent Integration of Information. *ACM Transactions on Information Systems*, Vol. 17(3), 270–293, 1999.
19. Gray, P.M.D, Preece, A., Fiddian, N.J. and colab. KRAFT: Knowledge Fusion from Distributed Databases and Knowledge Bases, *Proceedings of 8th International Workshop on database and Expert Systems Applications* (DEXA'97). 1997.
20. Gruber, T. A translation approach to portable ontology specifications. *Knowledge Acquisition 1993* –Vol.5(2), 199–220, 1993.
21. Hsu, C. and Knoblock, C. A. *Using Inductive learning to generate rules for semantic query optimization*. In Piatetsky-Shapiro, G., Fayyad, U., Symyth, P. and Uthurusamy, R. editors, Advances in Knowledge Discovery and Data Mining, chapter 17. AAAI Press, 1996.
22. Kakas, A. C., Kowalski, R. A., and Toni, F. .Abductive Logic Programming,. *Journal of Logic and Computation*. Vol.2(6), 719-770, 1993.
23. Kifer, M., Lausen, G., and Wu, J. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, Vol.42(4), 741–843, 1995.
24. Kitchenham, B. DESMET: A method for evaluating Software Engineering methods and tools Technical Report TR96-09. *Department of Computer Science, University of Keele, Staffordshire.* August 1996.
25. Lange, D., Oshima, M. *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley, 1998.
26. Lassila O., R.R. Swick,. Resource Description Framework (RDF) Modeland Syntax Specification. *W3C Recommendation*, February 22, 1999.
27. MacGregor, R. Inside the LOOM classifier. *SIGART bulletin*. Nº 2(3), 70-76, 1991.
28. MCC Carnot Project, http://www.mcc.com/projects/carnot
29. Medcraft, P., Schiel, U., Baptista, P. DIA: Data Integration Using Agents. Databases and Information Systems Integration. *ICEIS*, 79-86, 2003.
30. Mena, E., Kashyap, V., Sheth, A. and Illarramendi, A. Observer: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. *Kluwer Academic Publishers*, Boston. http://citeseer.nj.nec.com/mena96observer.html, 1-49, 2000.
31. Mena, E., Kashyap, V., Sheth, A. and Illarramendi, A. Managing Multiple Information Sources through Ontologies: Relationship between Vocabulary Heterogeneity and Loss of Information. *In Proceedings of Knowledge Representation Meets Databases (KRDB'96)*, ECAI'96 conference, Budapest, Hungary, 50-52, 1996.
32. Nam, Y. & Wang, A. (2002). Metadata Integration Assistant Generator for Heterogeneous Distributed Databases. In Proceedings *International Conference on Ontologies, Databases, and Applications of Semantics for Large Scale Information Systems*, Irvine CA, 28-30.
33. Preece, A., Hui, K., Gray, P. KRAFT: Supporting Virtual Organisations through Knowledge Fusion. Artificial Intelligence for Electronic Commerce: *Papers from the AAAI-99 Workshop*, Technical Report WS-99-01, AAAI Press, 33-38, 1999.
34. Preece, A., Hui, K., Gray, A., Jones, D., Cui, Z. The KRAFT Architecture for Knowledge Fusion and Transformation. In *19th SGES International Conference on Knowledgebased Systems and Applied Artificial Intelligence (ES'99)*, Berlin. Springer.
35. Siegel, M. and Madnick, S. E. A metadata approach to resolving semantic conflicts. In Proceedings of the 17th Conference on Very Large Data Bases (Barcelona, Spain, Sept.). *VLDB Endowment 1991*, Berkeley, CA, 133–145, 1991.
36. Tzitzikas, Y., Spyratos, N., Constantopoulos, P. Mediators over Ontology-based Information Sources. *Proceedings of the Second International Conference on Web Information Systems Engineering (WISE'02)*.IEEE 2002.
37. Visser, P. R. S., Jones, D. M., Bench-Capon, T. J. M. and Shave, M. J. R. Assessing heterogeneity by classifying ontology mismatches. *In Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS'98)*, IOS Press, 148–162, 1998.
38. Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H. and Hübner, S. Ontology-based Integration of Information - A Survey of Existing Approaches. *In: Proceedings of IJCAI-01 Workshop: Ontologies and Information Sharing*, Seattle, WA, 108-117, 2001.
39. Woelk, D., P. Cannata, M. Huhns, W. Shen, and C. Tomlinson. Using Carnot for Enterprise Information Integration. *Second International Conf. Parallel and Distributed Information Systems*. January, 133-136, 1993.
40. Woelk, P., Kim, W. And Lee, W. Query Processing in Distributed ORION. International Conference on Extending Datadase Technology, 169-187, March, 1990.

41. Woelk, D. and C. Tomlinson. The InfoSleuth Project: Intelligent Search Management via Semantic Agents. *MCC Tech. Report*, Sept., 1994. Also available in the Electronic Proceedings of the Second World Wide Web Conference '94 at http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/Searching/woelk/woelk.html.
42. Yang,H. Cui, Z. and OBrien, P., Extracting Ontologies from Legacy Systems for Understanding and Re-engineering. *IEEE International Conference on Computer Software and Applications*, 1999.
43. Zaniolo, C. The Logical Data Language (LDL): An Integrated Approach to Logic and Databases. *MCC Technical Report STP-LD-328-91*, 1991.

Combining Symbolic Execution and Model Checking to Reduce Dynamic Program Analysis Overhead

Néstor Catano *

Abstract

This paper addresses the problem of reducing the runtime monitoring overhead for programs where fine-grained monitoring of events is required. To this end we complement model checking techniques with symbolic reasoning methods and show that, under certain circumstances, code fragments do not affect the validity of underlying properties. We consider safety properties given as regular expressions on events generated by the program. Further, we show how our framework can be extended to consider programs with cycles. We sample our presentation with the aid of the Java PathFinder model checker [13].

Keywords: model checking, Java PathFinder, symbolic reasoning, instrumentation, monitoring, invariant strengthening.

1 Introduction

Testing is a method to check the satisfaction of a property in an implementation by means of experimentation. In testing, test cases are designed following what the experience of programmers suggests. Unlike other more formal techniques such as model checking and theorem proving, testing is not complete in the sense that it can only prove the presence of errors but not their absence.

When doing testing, in general it is not possible to cover all the possible cases for which a program could produce an error, since that would imply, at least, considering a case for each possible value of each variable, but variable domains are usually infinite.

Model checking [1, 2] is a verification technique especially conceived to prove properties of reactive systems. When model checking, ideally a user would not need to interact with the model checker at all; the user's main work would consist in pressing the model checker "go-ahead" button, waiting a couple of seconds, and finally, analyzing the result produced by the model checker. In practice however, model checkers need human interaction. Additionally, model checking techniques do not scale well for real-life problems because of the the state explosion problem.

Symbolic reasoning [10] arises as a means to supplement testing and model checking. To supplement testing because, when reasoning symbolically, variables are not constrained to a concrete value but are supposed to have generic symbolic values. Hence, it is easier to cover more test cases. Symbolic reasoning supplements model checking techniques as well, because an explicit-state model checker can be used, for example, to explore symbolic trees. Model checking can thus be used more effectively to prove (as opposed to refuting) properties.

In the following, we formulate the problem which this paper is concerned with. We will then describe how model checking techniques, enhanced with symbolic reasoning, can be used to address the problem.

*Department of Computer Sciences, The University of York, U.K. catano@cs.york.ac.uk

The problem. We consider systems composed of two components. The first component, a Java program in execution, is "monitored" by the second, an external observer. Monitoring consists in tracking variables and their values. The observer is given as a finite-state automaton which verifies properties expressed as regular expressions on events generated by the program. In order for the observer to verify the property, the Java program needs to be instrumented to transmit variables and values. This instrumentation basically consists of many communication instructions. Since such emission (communication) instructions negatively affect the performance of the program, one is faced with the problem of reducing as many useless emissions as possible. An emission is considered to be useless when it does not modify the semantics of the observer. The semantics of the observer is expressed as "the ability of transition under the same program conditions".

Here we only consider safety properties. For example, one might like to monitor that "the temperature never exceeds 100 degrees", where the temperature is given by variable `temp` in the program; hence, the value of `temp` will be emitted to the observer whenever `temp` is updated and the observer will make a transition when `(temp > 100)` is `true`.

This paper. We use model checking techniques to address the problem of reducing the number of such useless emissions for programs where fine-grained monitoring of events is required. We define fine-grained monitoring as those cases where many statements in the program can affect the observations, for example, in the case where the specific value of a variable is tracked. To monitor a certain variable `x`, instructions `emit(x)` must be introduced in the Java program in any place where `x` is modified; `emit(x)` communicates the variable `x` and its current value to the observer. Although we will focus our presentation on fine-grained monitoring, techniques proposed here will also work for larger-scale monitoring, but will not have the same degree of effectiveness.

We propose the use of model checking to show that for certain code fragments, under certain conditions, no emission will be required since it cannot affect the property that is checked. Specifically, we will show how to determine whether program statements will change the state of some observer, by doing a symbolic execution of the code within the Java PathFinder (JPF) model checker [13]. JPF has been recently extended with the capacity of doing symbolic reasoning [8], and it is this feature that we will use here to show under which conditions a program statement can change the state of an observer.

In some cases, model checking can show conclusively that a program will behave correctly according to some property, *i.e.* any execution of the program will be correct. However, model checking is in general much better at finding counterexamples than showing that a program is correct. Since our approach relies on the model checker being able to show that a property holds, rather than showing a refutation, we believe that symbolic reasoning is more appropriate than model checking based on explicit enumeration.

Since our approach essentially starts from the assumption that every statement potentially changes the state of the monitoring system, with model checking then used to reduce this number of observations, the scalability of model checking influences the accuracy and not the correctness of our approach. In other words, the more analysis is done with the model checker, the less runtime overhead during monitoring.

Contributions. This paper shows how model checking techniques — when supplemented with symbolic reasoning methods — can effectively be used to show that a certain property holds. The major weakness of the symbolic execution within JPF is that cycles cannot be handled in their full generality: when doing symbolic execution, JPF cannot determine whether a state has been revisited, and therefore the analysis will not terminate. To overcome this, we show how classical reasoning about loop invariants can be used within our framework to deal with cycles. Roughly speaking, when a loop invariant is known, emissions are added to the program only when they do not contradict the invariant. The other

emissions are useless and therefore should not be added.

Although, in general, finding loop invariants is an undecidable problem, we show how symbolic reasoning can be used to show that a property is a loop invariant. To do so, a loop invariant is conjectured — it is assumed at the beginning of the loop-body, and then checked immediately before its end — and successively refined, based on the information given by the symbolic execution of the loop.

The rest of this chapter. The rest of the chapter is structured as follows. Section 2 formalizes the problem of removing useless emissions when instrumenting programs in our framework. Section 3 introduces symbolic execution of programs. Ideas presented in this section provide a theoretical basis for a better understanding of subsequent sections. Section 4 gives an overview of the symbolic execution framework built on top of JPF. Section 5 presents how model checking with symbolic reasoning can be used to show the existence of useless emissions of non-looping programs. Section 6 extends Section 5 to consider loops. Lastly, Section 7 concludes and compares to related work.

2 Semantics of the observer

When running, a program generates events that can produce changes in the current state of the observer. We are interested in fine-grained monitoring, so any change of a specific variable value can potentially affect the semantics of (the current state of) the observer. Hence, only if each variable is tracked and its value sent to the observer (who will lastly check the property), the monitoring will be effective. To achieve an effective tracking, we instrument programs in such a way that emission statements are added in those parts of the code where modifications of variables are produced. These emission statements transmit the current value of the involved variables.

Emissions are "expensive", so having many of them will affect the performance of the program. We wish to remove as many useless emissions as possible. Useless emissions are those that, regardless of the transmitted values, will not make the observer transition.

To be tracked, instructions must be associated to locations. This association is only possible for non-looping programs. For looping programs this association cannot be done because in the general case the number of loop iterations cannot be decided without executing the loop itself. Section 6 presents our approach in dealing with loops. The rest of this section defines in detail the semantics of the observer.

Instructions and program locations. We call `i(loc)` the program instruction occurring at the program location `loc`. In the program below, method `m` declares a sole variable `x` which is then increased four times. Locations have been added into the program as comments; `i(0)` makes allusion to the instruction declaring the variable `x`, and successive instructions increasing variable `x` are referred to as `i(1)` through `i(4)`. The set of program locations is called `L`.

```
static void m() {
 0: int x = random();
 1: x++;
 2: x++;
 3: x++;
 4: x++;
}
```

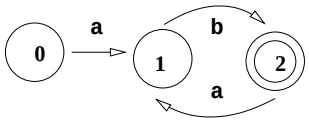


Figure 1: Automaton for $(\mathtt{ab})^+$

Instrumentation. Program instrumentation is accomplished by adding a single emission statement $\mathtt{emit}(\mathtt{loc}, \overrightarrow{\mathtt{x}})$ after each instruction $\mathtt{i}(\mathtt{loc})$, where $\overrightarrow{\mathtt{x}}$ represents the set of variables involved in the execution of the instruction. Emission $\mathtt{emit}(\mathtt{loc}, \overrightarrow{\mathtt{x}})$ sends the observer each variable in $\overrightarrow{\mathtt{x}}$ as well as its value.

Event generation. When a program is executed, it generates events that can make the observer transition. We define events generated by programs as having type $\mathtt{Ep} : \mathtt{Vr} \times \mathtt{Vl}$ between variables $\mathtt{Vr}$ and values $\mathtt{Vl}$, with the intuitive meaning "if one is interested in events as described by the relational operator $\mathtt{Ep}$, and after the execution of a certain instruction the value of variable $\mathtt{x}$ becomes $\mathtt{v}$, then the event as described by $\mathtt{Ep(x,v)}$[1] is produced by the execution of the instruction".

The event generation relation $\mathtt{Evt} : \mathtt{L} \to \mathcal{P}(\mathtt{Ep})$ associates a location $\mathtt{loc}$ with the set of events the instruction at location $\mathtt{loc}$ is able to produce. In our program, the event generation relation $\mathtt{Evt}$ depends on the value taken by $\mathtt{x}$ in its declaration. For instance, if $\mathtt{x}$'s initial value is 0 then $\mathtt{Evt}(0) = \{\mathtt{x} = 0\}$, $\mathtt{Evt}(1) = \{\mathtt{x} = 1\}$, $\mathtt{Evt}(2) = \{\mathtt{x} = 2\}$, $\mathtt{Evt}(3) = \{\mathtt{x} = 3\}$ and $\mathtt{Evt}(4) = \{\mathtt{x} = 4\}$.

Observer. We consider safety properties only, which are given as regular expressions over events tracked by the observer. Due to the relation between regular expressions and finite-state automata we chose these last as a model for the observer.

The observer is represented by the automaton $\mathcal{A} = (\mathtt{Q}, \mathtt{F}, \mathtt{q}_0, \mathtt{E}, \delta, \mathtt{Mp})$, where $\mathtt{Q}$ is the set of states of the automaton, $\mathtt{F}$ its set of final states, $\mathtt{q}_0$ its initial state, $\mathtt{E}$ its alphabet (of events), and $\delta : (\mathtt{Q} \times \mathtt{E}) \to \mathtt{Q}$ its transition relation. Instructions in the program generate a spectrum of events that should be mapped into words as understood by the automaton. Therefore, the automaton $\mathcal{A}$ is provided with a mapping relation $\mathtt{Mp} : \mathtt{Ep} \to \mathtt{E}$.

As an example, the automaton observer in Figure 1 is derived from the regular property $(\mathtt{ab})^+$, and its alphabet of events $\mathtt{E}$ is the set $\{\mathtt{a},\mathtt{b}\}$. Transitions for events other than $\mathtt{a}$ and $\mathtt{b}$ are undefined, *i.e.* they are supposed to be going into a certain trapping state. A mapping relation $\mathtt{Mp}$ for this automaton might, for instance, associate $\mathtt{x{=}1}$ in the program to event $\mathtt{a}$ in the observer; likewise $\mathtt{x{=}3}$ to $\mathtt{b}$.

Semantics. We define two relations on an automaton $\mathcal{A}{=}(\mathtt{Q},\mathtt{F},\mathtt{q}_0,\mathtt{E},\delta,\mathtt{Mp})$. The reaction relation $\mathtt{React} : \mathtt{L} \times \mathtt{Q} \to \mathcal{P}(\mathtt{E})$ relates a location $\mathtt{l}$ in the program and a state $\mathtt{q}$ in the automaton to the set of events $\mathtt{e} \in \mathtt{Evt}(\mathtt{l})$ for which $\delta(\mathtt{q},\mathtt{e}) \neq \mathtt{q}$. Also, relation $\mathtt{Stay} : \mathtt{L} \times \mathtt{Q} \to \mathcal{P}(\mathtt{E})$ is defined as the $\mathtt{React}$ complement relation, found when considering elements $\mathtt{e} \in \mathtt{Evt}(\mathtt{loc})$ for which $\delta(\mathtt{q},\mathtt{e}) = \mathtt{q}$ or $\delta(\mathtt{q},\mathtt{e})$ is *undefined*.

If variable $\mathtt{x}$ is initialized to 0, and $\mathtt{Mp}$ is the same mapping as before, the automaton will react (make a transition) with event $\mathtt{a}$ each time the current state of the automaton is 2 and the program is at location 1 — $\mathtt{React(1,2)(a)}$[2] holds. In contrast, when the automaton is at state 1 and the program at location 1 no reaction will be produced, *i.e.* $\mathtt{Stay(1,1)(a)}$ holds.

[1] Henceforth the more comfortable infix notation $\mathtt{x\,Ep\,v}$ will be used instead.

[2] Notice that sets are just predicates, so $\mathtt{s} \in \mathtt{S}$ is equivalent to $\mathtt{S(s)}$.

Procedure	::=	`procedure` Id`(`$\overrightarrow{param}$`)` Body `endp;`
Body	::=	DeclS StmtS
DeclS	::=	Decl DeclS
	|	Decl
Decl	::=	`declare` Vars `:` Type `;`
	|	`;`
StmtS	::=	StmtStmtS
	|	Stmt
Stmt	::=	Assig
	|	IfStmt
	|	WhileStmt
	|	ProcInv
	|	Return
	|	`;`
Assig	::=	Var `:=` Exp
IfStmt	::=	`if(`Cond`)` Body$_1$ `else` Body$_2$
WhileStmt	::=	`while(`Cond`)` `do` Body
ProcInv	::=	Id`(`$\overrightarrow{actual}$`);`
Return	::=	`return(`Exp`)`
	|	`return`

Figure 2: A simple Java imperative language

For any location $\mathtt{l} \in \mathtt{L}$, if for all $\mathtt{q} \in \mathtt{Q}$ and for all $\mathtt{e} \in \mathtt{E}$ $\mathtt{Stay}(\mathtt{l}, \mathtt{q})(\mathtt{e})$ holds, then emission at location `l` is useless. We want to remove all those useless emissions. We propose the use of model checking and symbolic execution to show that under certain conditions these removals are always possible.

Syntax for our programs. Figure 2 introduces the syntax for our programs; this syntax is defined in a Java-like style. In the declaration of a procedure (nonterminal Procedure), TypeRtrn is the type of the expression returned by the procedure (a `boolean`, an `integer` or the special symbol `void` meaning no value), Id the name of the procedure and $\overrightarrow{param}$ the list of its parameters. As usual in Java-like programs, after declaring the procedure signature, one goes on to the variable declaration section (Decl in rule for Body) and then continues writing the procedure statements (nonterminal StmtS). Those declared variables are exclusively bounded to the Body of the procedure.

A statement Stmt can take the form of a variable assignment, an `if` statement, a `while` statement, a procedure invocation or simply a skip instruction "`;`", doing nothing. An assignment such as:

Var := Exp

assigns the value of the expression Exp to the variable Var. Integer expressions are formed with the aid of the usual binary arithmetic operators `+`, `-`, `*` and `/`, and the unary arithmetic

operator `-`. Boolean expressions Cond are constructed from Boolean constants `true` and `false`, and from arithmetic expressions connected by relational operators `=`, `!=`, `<`, `<=`, `>`, or `>=`, and logical operators `&` (and), `|` (or), `!` (negation) and `=>` (implies).

An `if` statement as the following:

$$\texttt{if(Cond) Body}_1\ \texttt{else Body}_2$$

executes either $\texttt{Body}_1$ or $\texttt{Body}_2$ depending on the truth value of Cond. Finally, the procedure invocation:

$$\texttt{Id(}\overrightarrow{actual}\texttt{);}$$

causes the procedure Id to be invoked with parameters $\overrightarrow{actual}$. Formal parameters $\overrightarrow{param}$ in the declaration of the procedure Id are replaced by actual parameters, *i.e.* $[\overrightarrow{actual}/\overrightarrow{param}]$, using the Java convention.

3 Symbolic execution of programs

The symbolic execution of a program goes through symbolic states. A symbolic state is a tuple $\texttt{(x}_i\texttt{:X}_i\texttt{;pc:Bool)}$ composed of symbolic variables $\texttt{X}_i$ for variables $\texttt{x}_i$, and a so-called *path condition* `pc`. The path condition is a quantifier-free boolean formula over symbolic inputs, that accumulates constraints which inputs must satisfy in order (for an execution) to follow the particular associated path. The initial path condition, *i.e.* the path condition of the initial symbolic state, is `true` for any program. The path condition is updated as more program instructions are executed. Further, a path condition is not allowed to be `false` (an unreachable path). The meaning of the symbolic execution of programs is defined as follows:

(*i.*) *Symbolic value of expressions.* Given `x:X` and `y:Y` in some symbolic state, the symbolic value of the expression `x op y` is `X op Y`, where `op` is any of `+`, `-`, etc.

(*ii.*) *Symbolic execution of assignments.* The symbolic execution of an assignment `x:=` Exp updates the symbolic state `(x:X;pc)` to the state `(x:Symb(Exp),pc)`, where `Symb(Exp)` is the symbolic evaluation of expression Exp as described by Item *(i.)*.

(*iii.*) *Symbolic execution of conditional statements.* The symbolic execution of a conditional statement $\texttt{if(Cond) Body}_1\ \texttt{else Body}_2$ is accomplished according to the following steps. *(a.)* First, evaluate the boolean expression Cond[3], *(b.)* If the current path condition `pc` implies Cond, then no new sub-cases is necessary because `pc` already contains enough information to deduce that $\texttt{Body}_1$ must be executed. If `pc` implies `!Cond` then similarly the path condition does not require to be updated because it already contains enough information to deduce that $\texttt{Body}_2$ must be executed, else *(c.)* Establish a sub-case where the path condition of the current symbolic state is changed from `pc` to `pc & Cond`, then proceed to symbolically execute $\texttt{Body}_1$, and *(d.)* Establish a sub-case where the path condition of the current path condition is changed from `pc` to `pc & !Cond`, then proceed to symbolically execute $\texttt{Body}_2$.

(*iv.*) *Symbolic execution of annotations.* To symbolically execute the input annotation `assume` Exp `;` first evaluate Exp, *i.e.* `Symb(Exp)`, and then update the path condition `pc` to `pc & Symb(Exp)`. For output annotations `assert` Exp `;`, first evaluate Exp. And, if `pc` implies `Symb(Exp)` then the program is correct, otherwise the program fails.

[3]Note that this requires counting on a decision procedure. We will not go further into this topic here; we just assume that this decision procedure exists.

4 Symbolic execution in Java PathFinder

Our symbolic execution-based framework uses the Java PathFinder model checker (JPF) [13]. JPF is an explicit-state model checker for Java programs built on top of a custom-made Java Virtual Machine (JVM). JPF can handle all the language features of Java, and additionally it treats non-deterministic choice expressed in annotations of the program being analyzed. For symbolic execution, the JPF model checker has been extended to allow backtracking whenever a path condition is unsatisfiable. To determine the satisfiability of a formula, JPF calls a decision procedure provided by the Omega library [12]. In particular, an annotation `ignoreIf(cond)` is used to allow backtracking: whenever `cond` evaluates to `true` the model checker will stop exploring the branch and backtrack. This feature will also be used in the discovery of loop invariants in Section 6.

The main idea behind symbolic execution [10] is to use symbolic values, instead of actual data, as input values, and to represent the values of program variables as symbolic expressions. The state of a symbolically executed program includes, in addition to the symbolic values of program variables, the program counter and a path condition. The path condition is a quantifier-free boolean formula over the symbolic inputs; it accumulates constraints which the inputs must satisfy in order for an execution to follow the particular associated path. A symbolic execution tree characterizes execution paths followed during the symbolic execution of a program. The nodes represent program states and the arcs represent transitions between states.

As an example (taken from [8]), consider the code fragment in Figure 3, which swaps the values of integer variables `x` and `y`, when `x` is greater than `y`. Figure 3 also shows the corresponding symbolic execution tree. Initially, the path condition, `PC`, is `true` and `x` and `y` have symbolic values `X` and `Y`, respectively. At each branch point, `PC` is updated with assumptions about the inputs according to the possible alternative paths. For example, after the execution of the first statement, both `then` and `else` alternatives of the `if` statement are possible, and `PC` is updated accordingly. If the path condition becomes `false`, *i.e.* no set of inputs satisfy it, this means that the symbolic state is not reachable, and the symbolic execution does not continue on that path. For example, statement `(6)` is unreachable.

Symbolic execution techniques have traditionally come up in the context of sequential programs with a fixed number of integer variables. In [13], these techniques have been extended to handle dynamically allocated data structures (lists and trees), complex preconditions (disallowing cyclic lists), other primitive data (strings), and concurrency. A key feature of the algorithm implemented in JPF is that it starts the symbolic execution of a procedure on *uninitialized* inputs and uses *lazy initialization* to assign values to these inputs. Consequently, the parameters are initialized when they are first accessed during the symbolic execution of the procedure. This allows symbolic execution of procedures without requiring an *a priori* bound on the number of input objects. Procedure preconditions are used to initialize inputs with valid values only.

Recursion. The JPF algorithm implementation exploits the model checker's search capabilities to handle arbitrary program control flow. In the implementation, no requirement on the model checker to perform state matching is done since state matching is undecidable when states represent path conditions on unbound data. Furthermore, the symbolic execution of looping programs can explore infinite execution trees. To overcome this, Section 6 describes how to address cycles in Java PathFinder.

```
    int x, y;
    read x,y;
1:  if (x > y) {
2:    x = x + y;
3:    y = x - y;
4:    x = x - y;
5:    if (x > y)
6:     assert(false);
    }
```

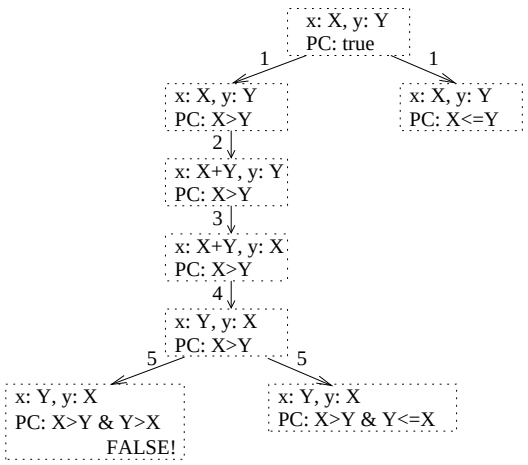


Figure 3: Code for swapping integers and corresponding symbolic execution tree.

5 Eliminating useless emissions for **Java** sequential programs

We use model checking and symbolic reasoning to show that we can remove useless emissions from a program and still *preserve* the semantics of the observer. The observer's semantics is defined as "the ability of reacting under the same program conditions", and formalized by the predicate `React` in Section 2.

Consider the instrumented program below, where an emission statement has been added into the program immediately after each modification of variable `x`, the only program variable. Also, consider the observer for the property $(\texttt{ab})^+$ in Figure 4, and the event mapping relation `Mp` which associates the event `x=1` in the program to the event `a` in the automaton, and `x=3` to `b`. The automaton in Figure 4 only reacts when variable `x` is 1 or 3. Variable `x` is initially given a nonnegative value as returned by function `random`; after the fourth increment of `x`, the value will be greater or equal than 4. Hence, the last emission will produce no reaction in the automaton, *i.e.* `Stay(4,q)(a)` for any $\texttt{q} \in \texttt{Q}$. Therefore, this emission becomes useless and it can be removed from the instrumented program.

```
static void m() {
 0: int x = random(); emit(0,x);

 1: x++; emit(1,x);
 2: x++; emit(2,x);
 3: x++; emit(3,x);
 4: x++; emit(4,x);
}
```

Below we present how the symbolic version of the program above is first elaborated and then model checked in the symbolic execution framework of JPF. We show that the last emission can be removed from the instrumented program, while preserving the semantics of the observer. Variable `x` of type `int` has been replaced by variable `X` of type `SymbolicInteger`, an integer implementation for `Expression`. Classes `SymbolicInteger` and `Expression` are defined in the module for symbolic execution built on top of JPF [8].

```
static void m() {
 0: Expression X = new SymbolicInteger();
    _addDet(GE,X,0);
```

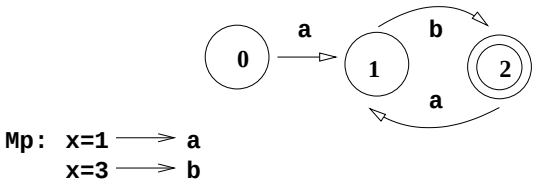


Figure 4: Mapping for `(ab)+`

```
    Emit(0,X);

 1: X = X._plus(1); Emit(1,X);
 2: X = X._plus(1); Emit(2,X);
 3: X = X._plus(1); Emit(3,X);
 4: X = X._plus(1); Emit(4,X);
}
```

After `X` is created, the condition `_addDet(GE,X,0)` saying that "`X`'s values are always greater or equal than 0" is added[4]. Moreover, operations increasing `x` have been replaced by method calls to `X`, *i.e.* `X._plus(1)`.

We also need to create a symbolic version for emissions. This symbolic version takes into account the way the automaton evolves from a particular state to another when the program is executed. Since each emission is associated with a location, the symbolic version `Emit` must be parameterized by the location.

The coding below presents the symbolic emission function `Emit` as it is implemented in JPF. The variable `state` in the guard of the `if` statement represents the automaton's current state, and the `_add` instructions reflect the behavior of the automaton's transitions. When executing the JPF model checker on the whole symbolic program, and once the conditions on the guard of the `if` statement are verified, the new conditions on `X` are added to the path condition. These `_add` instructions encode all possible reactions of the automaton. Hence, when the automaton stays in the same state, no new condition on `X` is added. Method `simplify` returns the path condition on `X` necessary for reaching the program location `loc` from the starting condition `x=x_init`[5]. Finally, method `changeState` updates the current state of the automaton according to the current path condition on `X`.

```
static void Emit(int loc,Expression X) {
 if(((state==0)&_add(EQ,X,1)) ||
    ((state==1)&_add(EQ,X,3)) ||
    ((state==2)&_add(EQ,X,1))) {
   simplify(Expression.pc,loc,X);
   changeState(X);
 }
}
```

When the whole program is symbolically executed in Java PathFinder, the path conditions `x=1`, `x=0`, `x=1`, `x=0` and `false` for respective locations 0 to 4 are yielded. From the last condition we conclude that the last emission will produce no reaction on the automaton for any `x`'s initial value. Therefore, that emission can be removed, while the semantics of the observer stays unchanged.

We use all those conditions on `x` not only for removing the last emission but for executing the other emission statements under the conditions yielded at each respective location. The

[4]This reflects the fact that `random` in the original program always returns a nonnegative integer value.
[5]`x_init` is the initial value returned by `random`.

final instrumentation for method `m` is presented below, where the previous conditions on variable `x` have been integrated into the original program. The last emission will never happen. Notice that all these conditions refer to the initial value `x_init` of `x`.

```
public static void m() {
 int x = random();
 int x_init = x;

 if (x_init == 1) emit(x);

 x++; if (x_init == 0) emit(x);
 x++; if (x_init == 1) emit(x);
 x++; if (x_init == 0) emit(x);
 x++; if (false) emit(x);
}
```

Similar considerations can be made when dealing with the whole syntax presented in Figure 2, except for loops. Therefore, Section 6 extends the work presented in this section to consider loops.

6 Eliminating useless emissions in cycles

When considering cycles, a similar analysis as in Section 5 cannot be done: when doing symbolic execution, the Java PathFinder model checker cannot determine whether a state has been previously visited; also, because the number of loop iterations cannot be determined beforehand, no full mapping between locations inside the loop and emission statements can be performed.

Hence, we should do a clever analysis of the information at hand. For instance, if we know the loop invariant, we can decide to emit only in those cases when the conditions under which emissions will occur do not contradict the invariant. However, the main difficulty is that finding loop invariants is an undecidable problem. One can also work with *partial* information, *i.e.* one can symbolically execute the program for a fixed number `n` of iterations, conjecture a *partial invariant*, and then remove any emission contradicting this partial invariant. Obviously, the intrinsic problem is that this partial invariant can be invalidated by a subsequent loop iteration.

In the following, we show how symbolic execution can be used to show that an initial conjectured invariant is a real invariant, and then, how to use this invariant information to remove emissions that produce no reaction in the automaton.

Guessing an initial conjectured invariant. First, a (symbolic) loop program is created for which all variables controlling the way the loop iterates are symbolic. Then, this (partially) symbolic program is executed and the path conditions generated at each loop iteration are checked. We conjecture a loop invariant based on the partial information at hand and proceed to prove that this conjecture is a real invariant.

We have rewritten "in a loop style" the example presented in Section 5 (see below). The only difference resides in the fact that we now have a random number `n` of assignments to `x` instead of a fixed number of 4 iterations. Furthermore, variable `i` has been introduced to control the current loop iteration.

```
public static void m() {
 int x = random();
 emit(0,x);
```

```
 int n = random();
 int i = 0;
 while(i < n) {
  i++;
  x++; emit(i,x);
 }
}
```

The program below is the symbolic on `X` version of the program above, while variables `i` and `n` remain concrete. We want to execute this symbolic program for different values of `n` in order to conjecture a loop invariant. We start from `n=6`.

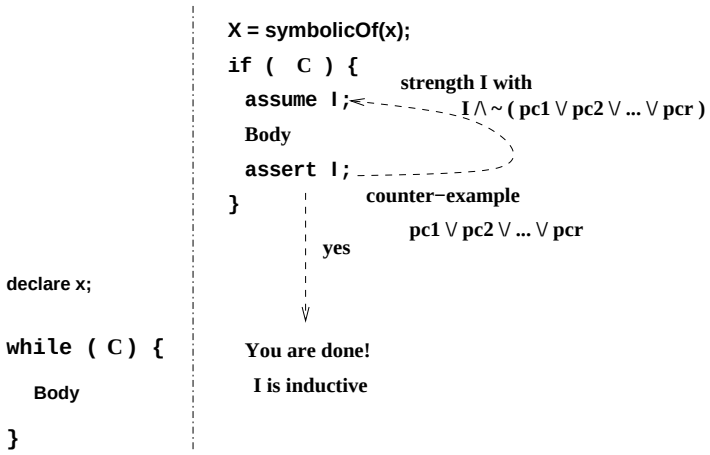
```
public static void m() {
 Expression X = new SymbolicInteger();
 _addDet(GE,X,0);
 Emit(0,X);

 int n = 6, i = 0;
 while(i < n) {
  i++;
  X = X._plus(1); Emit(1,X);
 }
}
```

When running JPF on this program, the path conditions `(x=1 || x=3)` for emission at `i=0`, `(x=0)` for emission at `i=1`, `(x=1 || x=2)` for emission at `i=2`, and `x=0` for `i=3` are established. Of the three remaining emissions for the three remaining iterations none is performed. Then, the number of iterations is increased to `n=13`. This time, the previous conditions for location `0` and for the six first iterations of the loop remain the same. No emission is performed for the other `7` iterations. Hence, we conjecture the invariant *"no emission is performed once* `i` *becomes greater than 3"* and proceed to prove whether it is a real invariant.

Proving the conjectured invariant. In general, symbolic execution of looping programs can produce *infinite* symbolic trees because another new path condition might be added each time the loop guard is visited. Hence, if symbolic techniques are to be used to prove that an initial conjecture I_k is a loop invariant, then the looping program must be made non-looping, *i.e.* the looping program must be *cut*. This non-looping program, however, must be general, in the sense that it must be representative of any symbolic execution visiting the loop guard. To achieve this generality, loop-guard variables must be declared symbolic, and then, when executed symbolically, their path condition initialized to the most general condition, namely, `true`.

The `if` program in Figure 5(b) is obtained when the symbolic tree of the `while` program in Figure 5(a) is cut. This new program provides a convenient way to show that the initially conjectured property I_k is an invariant. To show that I_k is an invariant, I_k must be *assumed* immediately after the guard, and then *asserted* immediately before the end of the body. If the evaluation of that assertion succeeds, then I_k is inductive. Otherwise, if the evaluation fails, the information given by the symbolic program is employed to strengthen I_k: if the execution of `assert` I_k`;` fails producing path conditions $pc_1 \vee \ldots \vee pc_k$, then I_k is strengthened to $I_{k+1} = I_k \wedge \neg(pc_1 \vee \ldots \vee pc_k)$, and the whole process is restarted; this time from I_{k+1}.

Below we present the full symbolic loop program produced when JPF is used through this iterative process of strengthening and checking. From Lines `2:` to `4:` symbolic variables for every concrete variable — including those concerned with the loop iterations — are created. From Lines `5:` to `7:`, initial conditions for these symbolic variables are added, *e.g* "`X` is



(a) `while` loop

(b) Iterative invariant strengthening

Figure 5: Loop invariant strengthening

nonnegative". When an `if` statement is executed symbolically, the guard must be asserted as a precondition to the execution of the loop body, Line `15:`. Also, the initial conjecture has been incorporated as a necessary condition to emitting, Line `18:`. Each one of Lines `9:` to `13:` represents an iteration refinement of the initial conjecture. Each one of these refinements are again checked after the loop body has been executed, Lines `20:` to `25:`. The last iteration produces no path condition invalidating the precedent strengthened $\mathtt{I}_k$. Hence, the loop invariant is the conjunction between the initial conjecture and each refinement.

```
 1:public static void m() {
 2: Expression X = new SymbolicInteger(),
 3:            N = new SymbolicInteger(),
 4:            I = new SymbolicInteger();
 5: _addDet(GE,X,0);
 6: _addDet(LT,I,N);
 7: _addDet(GT,I,0);

 9: ignoreIf(_add(GE,I,3)&_add(EQ,X,0));
10: ignoreIf(_add(GE,I,3)&_add(EQ,X,2));
11: ignoreIf(_add(GE,I,2)&_add(EQ,X,1));
12: ignoreIf(_add(EQ,I,1)&_add(EQ,X,0));
13: ignoreIf(_add(EQ,I,2)&_add(EQ,X,0));

15: _addDet(LT,I,N);
16: I = I._plus(1);
17: X = X._plus(1);
18: if (_add(GT,I,3)) assert(false);

20: if((_add(GE,I,3)&_add(EQ,X,0)) ||
21:    (_add(GE,I,3)&_add(EQ,X,2)) ||
22:    (_add(GE,I,2)&_add(EQ,X,1)) ||
23:    (_add(EQ,I,1)&_add(EQ,X,0)) ||
24:    (_add(EQ,I,2)&_add(EQ,X,0))
25:   ) assert(false);
```

```
26:}
```

Before being sure of whether this invariant is really inductive, a further condition must be checked. The invariant should be valid at the initial state. This condition must be checked in the instrumented program before the loop can be executed.

The instrumented program is shown below. From Lines `6:` to 8 the initial invariant condition is checked. For this particular example, this condition will always hold because the initial value for `i` is 0. In Line `12:`, the statement `emit(i,x)` is executed under the initial conjecture condition. Only when the condition evaluates to `true`, `x`'s value will be emitted. For the other cases we can not emit and still preserve the semantics of the observer, *i.e.* if `i>3`, then `Stay(i,q)(a)` and `Stay(i,q)(b)` for any `q` in the observer's set of states.

```
 1:public static void m() {
 2: int x = random(),
 3:     n = random(),
 4:     i = 0;

 6: if(((i>=2)&(x==0))||((i>=3)&(x==2))||
 7:    ((i>=2)&(x==1))||((i==1)&(x==0)))
 8:   println(``Invariant broken'');

10: while(i<n) {
11:  i++;
12:  x++; if(!(i>3)) emit(i,x);
13: }
14:}
```

7 Conclusion and related work

In this paper we showed that model checking techniques can be used to reduce dynamic program analysis overhead. Our approach basically consists in proving that certain code fragments will generate no reaction on the automaton-based monitoring system, and hence these code fragments can be removed. We developed our work in the framework of the Java PathFinder model checker (JPF), although the work is still valid in the framework of any model checker doing symbolic reasoning.

Our approach is general in the sense that there is no *a priori* restriction on the way the mapping from program events into events as understood by the observer is performed. However, when employing JPF to do symbolic execution, considered (automaton-based) properties cannot include transitions on the same state. This is not a drawback of our approach, but an aspect where the JPF model checker can be improved.

A drawback of the approach presented in this paper is that the refinement process is not always convergent and hence, it could happen that an invariant is never obtained, even though the invariant might exist. The whole process depends on the initial conjectured invariant and on the way each successive strengthened $\mathtt{I_{k+1}}$ is calculated.

C. Pasareanu and *W. Visser* in [11] propose an heuristic to achieve termination in this iterative process of strengthening. Yet, their problem is slightly different to that presented here. They are interested in showing that a loop program respects some property `P`[6]. They use an invariant strengthening algorithm similar to that described before, but additionally, at each step k of strengthening, the "exact" invariant I_k is newly strengthened iteratively. In this new strengthening process, the effect produced by the assertion of I_k at the end of the loop-body is not considered. If an error exists, then the method is guaranteed to terminate. If the program is correct with respect to the property, the method might not terminate.

[6]That is, `assert P;` is added immediately after the loop statement.

In [6, 7], *K. Havelund* and *G. Rosu* report on runtime verification in the framework of Java PathExplorer. It is composed of three main modules, namely, an instrumentation module, an interconnection module, and an observer module. The instrumentation module modifies the program byte codes so that relevant emissions are sent to the interconnection module, which in turn will retransmit these events to the observer module. The observer module checks for validity of temporal properties. Ideally, techniques to reduce program analysis overhead presented in this paper would serve as basis to enhance the runtime verification labor done by Java PathExplorer.

Two tools close to Java PathExplorer are Java-MaC (*M. Kim*, *S. Kannan*, *I. Lee* and *O. Sokolsky* in [9]) and DynaMICs (*A.Q. Gater*, *S. Roach*, *O. Mondragon* and *N. Delgado* in [5]). Java-MaC separates monitoring task from checking task. This separation makes Java-MaC an extensible open architecture. Unlike Java PathExplorer, in DynaMICs, properties targeted for verification are expressed as constraints. For instance, for the problem of the division of two integers `x` and `y`, yielding a quotient `q` and a reminder `r`, two constraints can be defined: `r < y` and `(q` $\times$ `y) + r = x`.

C. Flanagan and *Sh. Qadeer* in [4] present an abstraction-based method to infer loop invariants. They infer loop invariants to verify programs that manipulate unbound data such as arrays. Loop invariants for each loop are computed by iterative approximation. The problem of their approach is that it requires ingenuity in finding the initial property for iterative approximation.

T. Colcombet and *P. Fradet* in [3] propose a method to enforce trace properties. The programmer specifies the property `T` separately from the program `P`, and a "transformer" takes `T` and `P` and produces another equivalent program that satisfies the property. They only consider safety properties. An advantage of the work presented in this paper is that we consider not just "plain" events, but also values of variables and symbolic constraints over these variables.

As future work we plan to formalize and prove theorems to establish precisely the kind of programs that can be monitored with the methodology presented in this paper.

Acknowledgements. Thanks to Dr. Willem Visser for inviting me to NASA Ames in the summer of 2002, when this work was mainly carried out, and discussing topics on model checking and symbolic methods. Thanks to Dr. Corina Pasareanu with whom I had fruitful discussions on automatic loop-invariant generation.

References

[1] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit L. Petrucci, Ph. Schnoebelen, and P. Mackenzie. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer-Verlag, 1999.

[2] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 2000.

[3] T. Colcombet and P. Fradet. Enforcing trace properties by program transformation. In *Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 54–66, 2000.

[4] C. Flanagan and S. Qadeer. Predicate abstraction for software verification. In *Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 191–202. ACM Press, 2002.

[5] A.Q. Gates, S. Roach, O. Mondragón, and N. Delgado. DynaMICs: Comprehensive support for run-time monitoring. In K. Havelund and G. Rosu, editors, *Electronic Notes in Theoretical Computer Science*, volume 55. Elsevier, 2001.

[6] K. Havelund and G. Rosu. Java PathExplorer — a runtime verification tool. In *Proceedings of 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space, ISAIRAS'01*, Montreal, Canada, Jun. 18–22 2001.

[7] K. Havelund and G. Rosu. Synthesizing monitors for safety properties. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 342–356, 2002.

[8] S. Khurshid, C. Pasareanu, and W. Visser. Generalized symbolic execution for model checking and testing. In *Proceedings of TACAS03: Tools and Algorithms for the Construction and Analysis of Systems*, volume 2619 of *Lecture Notes in Computer Science*, Warsaw, Poland, Apr. 2003.

[9] M. Kim, S. Kannan, I. Lee, O. Sokolsky, and M. Viswanathan. Java-MaC: a run-time assurance tool for Java programs. In K. Havelund and G. Rosu, editors, *Electronic Notes in Theoretical Computer Science*, volume 55. Elsevier, 2001.

[10] J.C. King. Symbolic execution and program testing. *Communications of ACM*, 19(7):385–394, 1976.

[11] C. Pasareanu and W. Visser. Verification of Java programs using symbolic execution and invariant generation. In *Proceedings of Model Checking Software: 11th International SPIN Workshop*, Lecture Notes in Computer Science, Barcelona, Spain, Apr. 1-3 2004. Springer-Verlag.

[12] W. Pugh. The Omega test: A fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 31(8), Aug. 1992.

[13] W. Visser, K. Havelund, G. Brat, and S.J. Park. Model checking programs. In *Proceedings of the 15th IEEE International Conference on Automated Software Engineering*, Grenoble, France, Sept. 2000.

Algoritmo de emparejamiento de perfiles en Servicios Web Semánticos

José Javier Samper [*] Eduardo Carrillo[**] Juan José Martínez [*]

Resumen

Este artículo describe las principales características de un algoritmo de emparejamiento de Servicios Web Semánticos . El algoritmo aprovecha al máximo las capacidades proporcionadas por la ontología de descripción de servicios y constituye una mejora en relación con propuestas existentes. Además se describen los principales componentes relacionados con el proceso de implementación. El sistema desarrollado interactúa con ontologías de descripciones de conceptos desarrolladas en DAML+OIL y descripciones de servicios en DAML-S, usando como repositorio/razonador el sistema Sesame+BOR. Finalmente, se describe la implementación de comparaciones entre parámetros de perfiles de servicio mediante consultas realizadas a la base de conocimiento.

Palabras claves: *Web Semántica, Servicios Web, Servicios Web Semánticos, Emparejamiento*

Abstract

This paper describes the main characteristics of an algorithm for matching Semantic Web Services. The algorithm takes advantage of the capacities provided by ontologies that describe services and constitutes an improvement when compared to existing proposals. In addition, the main elements related with the implementation are also described. The system interacts with ontologies of descriptions of concepts developed in DAML+OIL and descriptions of services in DAML-S, using the reasoner and Sesame+BOR reasoner/repository. Finally, we explain the implementation of comparisons among parameters from service profiles by means of queries to the knowledge base.

Keywords*: S**emantic Web, Web Services, Semantic Web Services, Matchmaking.*

1 Introducción

Dos tecnologías fundamentales para el intercambio de información en la Web del mañana son la Web Semántica y los Servicios Web. La Web Semántica tiene como visión la búsqueda de una Web más inteligente en la que se pueda conseguir una comunicación efectiva entre ordenadores y centra sus esfuerzos principalmente en la búsqueda de descripciones enriquecidas semánticamente para los datos en la Web. En este sentido, se promueven descripciones que incluyan no sólo las estructuras de datos, sino también las relaciones existentes con otros conceptos, las restricciones, reglas que permitan realizar inferencia, etc. Así mismo se promueve la definición y reutilización de vocabularios u ontologías de conceptos que faciliten el procesamiento por parte de las máquinas. Por lo tanto, lenguajes descriptivos como DTDs, XML Esquemas y RDFS no son suficientes para

[*] Universidad de Valencia, Instituto de Robotica, Polígono de la Coma s/n, Paterna, Valencia, España.
[**] Universidad Autónoma de Bucaramanga,. Calle 48 Nro 39-234, Bucaramanga, Colombia.

describir las relaciones complejas existentes en ontologías, por lo que se han propuesto otros lenguajes como DAML+OIL y OWL que tienen mayor expresividad.

Por otra parte, los Servicios Web promueven la interacción entre *aplicaciones*. El servicio es un componente software que puede procesar un documento XML que recibe a través de combinaciones de protocolos de transporte y de aplicación. Los Servicios Web están diseñados para mover documentos XML entre procesos de servicio usando protocolos estándar de Internet.

Dos descripciones XML iguales de un servicio Web tienen un significado u otro dependiendo del contexto donde se encuentren, mientras que las descripciones semánticas cubren esta carencia. El uso de la semántica para describir servicios solventa este problema de los sistemas de emparejamiento basados en UDDI. A partir de las propuestas de la Web semántica se creó la ontología de orden superior DAML-S para descripción semántica de Servicios Web, que más tarde evolucionó en OWL-S (basadas en los lenguajes de marcado semántico DAML y OWL respectivamente) [4], las cuales permiten describir semánticamente las capacidades los Servicios Web, para que agentes software puedan leer de esas descripciones y razonar sobre la forma de interactuar con los servicios que ellas describen.

En la actualidad existen algunas propuestas de integración para emparejamiento de Servicios Web como las presentadas en [8], [18] y [5]. La arquitectura de integración descrita en [5] emplea técnicas de recuperación de información, Inteligencia Artificial e Ingeniería de Software para procesar la semejanza sintáctica y semántica entre descripciones de capacidad de servicios descritos en DAML-S. La arquitectura propuesta en este trabajo se basa en el uso de sistemas multiagente, en el que un agente denominado *matchmaker o emparejador* es el encargado de encontrar (de manera similar a un sistema de páginas amarillas) el servicio, simple o complejo que ha sido expuesto por un proveedor y que satisface las necesidades del cliente a partir de los parámetros definidos por éste.

Como aparece en [7] una descripción de servicio es una colección consistente por ella misma de restricciones sobre propiedades nombradas de un servicio. Los perfiles describen capacidades de servicio, por tanto pueden describir tanto las capacidades de los servicios ofertados por los proveedores (los anuncios) como lo esperado por los clientes (peticiones). Las peticiones son enviadas a registros de servicios de Web que los emparejan con perfiles de anuncios de otros servicios almacenados en alguna base de datos (repositorio) para identificar qué servicios proveen el mejor emparejamiento.

Por otra parte el proceso de *emparejar* consiste en podar el espacio de posibles emparejamientos entre posibles servicios ofertados y peticiones. Un servicio de emparejamiento requiere tanto metadatos ricos y flexibles como algoritmos de emparejamiento [15]. El emparejamiento entre anuncios y peticiones se considera idóneo cuando ambos son lo suficientemente similares. Un emparejamiento vendrá determinado por los diferentes grados de similitud.

Este artículo se basa en la propuesta de un algoritmo de emparejamiento de Servicios Web descritos semánticamente mediante comparaciones de parámetros que hacen uso de ontologías de conceptos y de servicios, como una extensión a propuestas actuales. Para conseguir este objetivo, el artículo ha sido organizado en la siguiente forma: en la sección 2 se presenta el estado del arte, en la sección 2.1 se describen las ontologías para descripción semántica de servicios y los sistemas de emparejamiento. Posteriormente en la sección 3 se describe el algoritmo propuesto, en la sección 4 la implementación de las consultas y en la sección 5 se describe el análisis de los resultados obtenidos.

2 Estado del arte

2.1 Ontologías para descripción semántica de servicios

OWL-S (o DAML-S) define 3 ontologías para la descripción, invocación y ejecución de servicios. La ontología Service Profile es usada para describir y anunciar el Servicio Web requerido por el usuario y ofertado por el proveedor respectivamente. La ontología Service Model es usada para definir el modelo de proceso y ejecución del servicio y nos sirve por tanto para saber cómo funciona, y Service Grounding es usada para describir cómo acceder al servicio, indicándonos cómo puede ser usado. OWL-S nos da lo necesario para conseguir la representación semántica de servicios basándose en OWL, que soporta razonamiento sobre inclusión en taxonomías de conceptos y permite la relación entre conceptos pudiendo expresar por ejemplo que X es parte de Y, o de forma más general, que existe una relación entre X e Y. Aunque tiene limitaciones, es suficientemente expresivo como para permitir la descripción de un gran número de servicios y permitir emparejamientos entre ellos.

2.2. Emparejamiento

En relación con el concepto de emparejamiento existen trabajos provenientes de la Ingeniería del Software, como la propuesta presentada por Zaremski y Wing en [20] y [21], relacionada con la reutilización de software y recuperación de librerías, mediante la especificación y posteriormente emparejamiento de componentes software. En [21] extienden el concepto de *signature matching* [20], para tener en cuenta las restricciones en las entradas y en las salidas en funciones y módulos. Este concepto es llamado *specification matching* y provee no sólo información de tipo estático sino también descripciones de comportamiento dinámico. Los autores distinguen varios tipos de emparejamientos en la especificación de emparejamientos de software.

En [17] Sycara et al. presentan LARKS, en el cual los servicios son vistos como frames y sus slots input, output, inConstraints y outConstraints pueden ser usados para describir los atributos esenciales de un servicio. Los conceptos usados por las descripciones pueden ser definidas mediante un lenguaje de descripción de conceptos denominado ITL (Information Terminological Language). Este lenguaje común es usado por los agentes mediadores para emparejar agentes de solicitud de servicio con agentes proveedores de servicios que resuelvan los requerimientos del agente que hace la solicitud.

El proceso general que se sigue en el emparejamiento de servicios ofertados y requeridos se basa en que cuando un agente envía una petición al *matchmaker* entonces más tarde, éste le devolverá como resultado información respecto al servicio que empareje con la descripción dada en el requerimiento. Esta información incluirá las IOPEs (*Inputs, Outputs, Preconditions, Effect*s) las cuales están reflejadas en el perfil del servicio así como información adicional de parámetros de servicio, información del proveedor etc., que el proveedor del servicio habrá comunicado con anterioridad al emparejador.

En la práctica el cliente debería ya conocer las IOPEs y lo que no sabrá es el tipo de proceso, si será atómico o por el contrario se tratará de una composición de servicios en cuyo caso habrá que realizar pasos intermedios que consultarán el modelo de proceso para identificar el flujo de trabajo desde la entrada a la salida (con la posibilidad de que la salida de un proceso intermediario sea usada como entrada en el siguiente proceso atómico en la secuencia dada) hasta que todos los procesos involucrados sean ejecutados. Por tanto los agentes que realizan la petición de servicio, deberán de ser capaces de interpretar el contenido de la ontología de proceso y el grounding para poder comunicarse con un agente proveedor de tal servicio [22] [10].

La idea principal que subyace en la aproximación de emparejamiento viene dada por la búsqueda de un servicio basada en el tópico o categoría y después en el uso de los parámetros de salida y resto de parámetros. El problema tal y como apuntaba Massimo Paolucci en [10] es que el uso de categorías de servicio puede no tener casi significado si las categorías permitidas en la ontología son demasiado amplias. Por otra parte, para dotar de significado se necesitará especificar muchas clases diferentes, una por cada tipo de servicio. El uso de los parámetros de entrada y salida permitirán solventar este problema, permitiendo una definición implícita de los servicios. Esto requerirá el uso de ontologías menos precisas aunque, por el contrario, serán necesarias

descripciones de servicios más esmeradas y un proceso de emparejamiento más complejo. La idea por tanto es soportar ambos tipos de emparejamiento, debido al resurgimiento de grandes ontologías y clasificaciones de servicios.

El modelo y algoritmo de emparejamiento semántico planteado en [13] es el que ha sido utilizado por la mayoría de investigadores como base para sus sistemas. Plantean un sistema de emparejamiento basado en la semántica descrita en el perfil (Profile) de los servicios y en la utilización de los registros UDDI para mantener las descripciones de los servicios, aunque el referente para el resto de investigadores ha sido el algoritmo de emparejamiento utilizado. Hacen uso de la ontología perfil del servicio y de DAML-S como lenguaje de especificación para las descripciones del servicio. Se asume que los servicios de la red anuncian sus interfaces (inputs/outputs) usando la misma ontología. En este trabajo se aborda la importancia para la clasificación del emparejamiento de las salidas. Un emparejamiento entre un anuncio y una petición de servicio consiste en emparejar todas las salidas de la petición con las del anuncio; y todas las entradas del anuncio con las de la petición. El de las entradas se usa más que nada para resolver las igualdades que se produzcan.

En [23] se expone una extensión del algoritmo de emparejamiento de Paolucci et al. En este caso el cliente define las precondiciones que piensa que debe tener el servicio para que emparejen con el mayor grado posible con las que los proveedores introducen en los perfiles (Profiles). Tuvieron en cuenta las precondiciones porque consideran que son unos parámetros importantes en las negociaciones, ya que puede darse el caso de Servicios Web que incluyan precondiciones que negocian el tipo de pago o el tipo de tarjetas de crédito que admiten etc.

Sin embargo, este tipo de emparejamiento no se ha tenido en cuenta en nuestra propuesta, por la peculiaridad de los servicios de información de tráfico[1], marco en el cual se decidió probar nuestra arquitectura.

2.2.1. Grados de similitud o match

El grado de similitud depende de la relación entre los conceptos (tomada de las ontologías) que se están comparando, y generalmente se reduce a la mínima distancia entre ellos en el árbol taxonómico. La función principal del algoritmo de emparejamiento que usamos, consiste en determinar el grado de similitud o diferencia entre descripciones de servicios, tomando como base el conjunto de relaciones entre ellos.

La denominación de los grados varía según la literatura [1],[9],[13],[19]:

- *Exact*: cuando los conceptos tanto en la petición como en el anuncio son equivalentes.
- *Subclass of:* determinado por la relación *"ser subclase de"*. Cuando los conceptos en la petición son subclase (relación directa) de los del anuncio. (En [13] incluyen como e*xact matching* a este tipo de emparejamiento).
- *Subsumption*: la cual puede ser de dos tipos:
  - Plug-in o Contained: cuando los conceptos en el anuncio A incluyen los de la petición P. Formalmente, (P ? A). En este caso, la petición puede ser satisfecha debido a que los conceptos del anuncio, son más generales que los de la petición, y por tanto existe la posibilidad de que el cliente pueda cumplir sus objetivos. Sin embargo, no se considera que exista una relación de subclase directa (grado anterior)

[1] En el caso de servicios que ofrezcan información de tráfico gratuita a un usuario, después de la ejecución del servicio, el usuario dispondrá del conocimiento requerido, pero este cambio de estado no es un cambio tangible y por lo tanto no queda muy claro la posibilidad de su uso en el proceso de descubrimiento de un servicio o en la composición, donde los efectos de unos servicios se conviertan más tarde en precondiciones de otros.

- Subsume o Container: cuando los conceptos en la petición incluyen los del anuncio; formalmente, (A ? P). Este tipo de emparejamiento no satisface completamente la petición pero puede ser considerado como una solución parcial válida ya que puede permitir al cliente que realizó la petición ir alcanzando parcialmente sus objetivos o metas.

? *Fail ,nul o Disjoint* cuando no hay relación de inclusión entre los conceptos; formalmente, (A ? P) ? ?

En [7] introdujeron nuevos tipos de emparejamiento que más tarde fueron adoptados por Li y Horrocks [9] como extensión a los anteriormente expuestos:

? *Intersection* u *Overlap*. Si la intersección de un anuncio A y una petición P se satisface, es decir son compatibles; formalmente, ? (A ? P ? ?).

Para entender el proceso de emparejamiento mencionado, es de suma importancia tener en consideración la definición de "*Open World descriptions*" aportada por Tommaso Di Noia et al. en [12]: *"La ausencia de una característica en la descripción de un anuncio o de una petición no se debe interpretar como una restricción de ésta. En su lugar, debe ser considerada como una característica que se podría refinar más tarde, o dejarla abierta si se considera irrelevante para el usuario"*. Esta definición clarifica la idea de que incluso cuando los servicios anunciados y los requeridos no tienen un emparejamiento exacto, puede ser posible o necesario usarlos en instancias específicas. Por tanto, los emparejamientos parciales también son importantes [19]. Lo anterior es conocido como "emparejamiento flexible" para distinguirlo del exacto considerado el más restrictivo de todos.

En [6] se establecen tres diferentes tipos de emparejamientos dados por las diferentes relaciones entre perfiles de petición y ofertados. Para expresarlo formalmente hacen uso de Lógica Descriptiva (DL). Siendo T una ontología común establecida para la descripción de servicios:

? Implicación: T ?? (P ? A) y T ?? (A ? P)[2]. Cada restricción impuesta por P es completada por A y viceversa. Da lugar al emparejamiento exacto.

? Consistencia: (A ? P) es satisfactoria en T, donde las restricciones no se excluyen mutuamente. En este tipo de emparejamientos es necesario establecer un límite en la distancia entre ambas descripciones, que se medirá teniendo en cuenta dicha ontología. Es decir, ¿cuántos destalles en P tengo que preguntar a la otra parte A?. Emparejamiento potencial.

? Inconsistencia: A ? P es insatisfactoria en T, lo que implica que algunas restricciones de una descripción están en conflicto con las de la otra. En este tipo de emparejamientos cabe preguntarse por el grado de inconsistencia de A en P, es decir ¿Cuántos detalles en P tengo que eliminar para poder aceptar A?. Emparejamiento parcial.

3 Algoritmo de emparejamiento propuesto

Para el desarrollo de este algoritmo se han tomado como base los siguientes interrogantes:

? ¿Qué parámetros de la clase Profile podrían ser usados para la búsqueda de servicios?

? ¿Qué grados de emparejamiento se deberían utilizar en las comparaciones de los parámetros funcionales?

[2] Extensión del concepto de equivalencia dado por la visión simplista de tratar únicamente la equivalencia sintáctica.

A partir del análisis de estos requisitos se ha desarrollado el algoritmo que se describirá en los siguientes apartados.

3.1 Algoritmo Base

El algoritmo consiste en buscar emparejamientos de valores semánticos que fueron utilizados para describir cada una de las capacidades del servicio, tanto por parte del proveedor como por parte del cliente en su petición. Por este motivo, las peticiones del cliente serán perfiles como los que utilizan los proveedores para anunciarse.

El algoritmo planteado en este artículo toma como base el algoritmo de Paolucci et al., utilizado en su sistema emparejador, pero posee algunas diferencias, las cuales se centran en cuatro aspectos:

1. Filtrado en función del perfil de la petición, de los perfiles de proveedores utilizados para el emparejamiento, para reducir el coste del algoritmo.
2. Uso de parámetros no funcionales[3].
3. Obtención del grado de emparejamiento mediante el uso de nuevos grados.
4. Mejoras en la ordenación de resultados.

3.1.1 Fases del algoritmo:

Las fases del algoritmo propuesto son:

- ***Fase 1***: Filtrar entre todos los anuncios de servicios, aquéllos que pertenecen a la misma categoría que la solicitada por el cliente:

El hecho de utilizar nuevos filtros nos permite aprovechar otras características que posee la clase *Profile* para anunciar Servicios Web y que no fueron explotadas por anteriores algoritmos.
El algoritmo de emparejamiento tiene un coste considerable ya que intenta emparejar la petición con cada uno de los proveedores disponibles, y para hacer esto cada una de sus entradas o salidas se contrasta con todas las de cada proveedor para encontrar la que más se le parezca semánticamente. Por esta razón, incluimos un filtrado anterior al emparejamiento, cuya finalidad es descartar proveedores que no cumplan unas determinadas características definidas también en los perfiles de la ontología.

El filtrado está basado en la categoría de servicio y es de carácter obligatorio en la formulación de la petición de servicio por parte del usuario. Servirá al sistema *emparejador* para obtener del repositorio de perfiles (*profiles)* aquella lista de servicios que pertenezcan a la categoría que busca el cliente.

- ***Fase 2***: De la lista resultante, combinar todas las posibles parejas entre la petición del cliente y cada uno de los anuncios de proveedores:

Esta fase consiste en establecer las distintas combinaciones posibles entre la petición dada por el cliente y los anuncios publicados por los proveedores, que por la fase anterior, pertenecen a la misma categoría de servicio.

---

[3] Son atributos que no son obligatorios. Suelen utilizarse para que los propios proveedores añadan las características especiales que creen que ofrecen sus servicios Web. Se pueden añadir más a los ya definidos gracias a otras propiedades definidas para la clase Profile, pero los principales son: ServiceCategory, QualityRating y GeographicRadius.

- ***Fase 3***: Cada vez que se obtiene una pareja en la fase anterior, aplicar sobre ella los diferentes grados de similitud para los parámetros funcionales y calcular los pesos relativos para:

i. Parámetros funcionales correspondientes a salidas,
ii. parámetros funcionales correspondientes a entradas y
iii. parámetros no funcionales

El cálculo de pesos relativos a las entradas y parámetros no funcionales (ii e iii) se hará siempre que éstos hayan sido especificados por el cliente y si el grado de similitud de una pareja en cuanto a sus salidas es positivo, es decir, si tienen alguna salida en común.
El riesgo de obtención de falsos positivos o negativos será menor dependiendo de la precisión en la asignación de pesos a las parejas "*petición cliente – anuncio proveedor"*

Los dos principales procesos que caracterizan esta fase son:

a) **Obtención del grado de emparejamiento para parámetros funcionales**:

En [13] se establece una función de ranking (*DegreeOfMatch*()) en la que se diferencian cuatro posibles casos además del de fallo. Además se le asigna el mismo peso (exact) no solo a conceptos iguales sino también al caso en que los conceptos de la petición son subclase (relación directa) de los del anuncio.
En [9] además de diferenciar los grados de Paolucci en cuanto a emparejamiento *exacto* y *ser subclase de*, destaca otro posible emparejamiento que ya fue anteriormente expuesto por [7]:

? Grado de *intersection* cuando un anuncio y una petición son compatibles, es decir mantienen algo en común.

Se propone un algoritmo con características similares a los aquí expuestos pero con importantes modificaciones en los grados de emparejamiento ya que en estos trabajos eran demasiado generales, por lo que se propone una versión compuesta por siete grados de emparejamiento que detallamos a continuación, en orden descendente de importancia:

- ***Exacto****:* los conceptos definidos por el cliente y por el proveedor son los mismos.
- ***CsubclaseP****:* dentro del árbol de la taxonomía de conceptos la distancia entre el concepto demandado por el cliente y el ofrecido por el proveedor es igual a 1, siendo por tanto, el descrito por el cliente, subclase directa del que proporcionó el proveedor. En este caso el proveedor ofrece un concepto más general que el que pide el cliente. El concepto del cliente es más restrictivo pero está incluido en el que proporciona el proveedor.
- ***PsubclaseC****:* el concepto descrito por el proveedor es subclase directa del que pide el cliente, es decir, el proveedor ofrece un concepto más restrictivo que el demandado por el cliente.
- ***PsubsumeC****:* el concepto descrito por el cliente se encuentra dentro del subárbol de conceptos descendiente del definido por el proveedor. Sería equivalente al *Plugin* definido en [13] aunque se diferencia de él en que no permite que el cliente especifique el nivel de profundidad máximo hasta el que llegará la búsqueda. Se ha optado por no permitir al cliente especificar esto porque consideramos que conceptos a distancias mayores o iguales a 3 niveles no tienen prácticamente relación semántica, debido al modo en que se construyen las jerarquías de conceptos. Por ello se ha decidido fijar la profundidad máxima en dos niveles, por lo que sólo se comprueba si el concepto definido por el cliente es como máximo "nieto" del concepto del proveedor. De no hacerlo así, aumentaría el riesgo de falsos positivos y por tanto, podríamos obtener servicios como válidos, cuando la relación semántica entre el concepto que se provee y el que pide el cliente es tan lejano semánticamente que no responde a las expectativas de éste.

- ***CsubsumeP:*** el concepto descrito por el proveedor se encuentra dentro del subárbol de conceptos descendiente del definido por el cliente, es decir, es el caso inverso al anterior, y es equivalente al grado *Subsume* definido en [13]. Como en el caso anterior, aquí también se limita a dos niveles de profundidad la comprobación de si el concepto demandado por el cliente incluye (*subsume*) al ofertado por el proveedor.
- ***ChermanoP:*** el concepto proporcionado por el cliente y el del proveedor tienen restricciones en común en alguna propiedad que ambos poseen, y además, tanto el concepto ofertado por proveedor como el demandado por el cliente son hijos del mismo padre, es decir, son conceptos "hermanos".
- ***Fail:*** cuando no se cumple ningún caso de los anteriores, es decir el concepto del proveedor y el de cliente no tienen relación alguna.

El motivo de este orden para los casos de subclase, es decir, porque CsubclaseP es mejor grado que PsubclaseC, es debido a que consideramos más importante que el proveedor ofrezca un producto menos restrictivo que el cliente, ya que en este caso puede suceder que también ofrezca lo que solicita el cliente. En caso contrario, si un proveedor ofrece algo más restrictivo que lo que solicitó el cliente, puede que a éste no le interese. El caso del orden entre PsubsumeC y CsubsumeP es análogo solo que considerando una mayor distancia en el árbol de la taxonomía de conceptos especificados en la ontología.

El pseudocódigo de nuestro método para la asignación de grados de emparejamiento se puede observar en la Figura 1:

```
DegreeOfMatch(outR,outA)
        {
                if Iguales(outA, outR)
                        return EXACTO
                else if SubClassOf(outR,outA)
                        return CSUBCLASEP
                else if SubClassOf(outA,outR)
                        return PSUBCLASEC
                else if Subsumes(outA,outR)
                        return PSUBSUMEC
                else if Subsumes(outR,outA)
                        return CSUBSUMEP
                else if Hermanos(outR,outA)
                        return CHERMANOP
                else
                        return FALLO
```

Fig. 1 Método de asignación de grados

b) Coincidencia exacta en parámetros no funcionales.

El algoritmo de Paolucci et al. no aprovecha otras propiedades de la descripción del perfil (*Profile)*, como son los parámetros no funcionales, que también aportan información semántica del servicio web, por lo que uno de los objetivos fue cubrir este vacío. El algoritmo emparejador explota al máximo las posibilidades de *Profile*, con el fin de hacer uso de todas las capacidades de los servicios, descritas semánticamente.

A continuación veamos qué filtros se han desarrollado para los diferentes parámetros no funcionales:

- ***filtro para región:*** con él se comprobará si el radio geográfico dado por el cliente es el mismo que el proporcionado por el proveedor.
- ***filtro para calidad de servicio:*** consiste en comprobar en el repositorio si la calidad que aportan los Servicios Web es la misma que la pedida por el cliente.

- ***filtro para nombre de servicio**:* en esta consulta se comprueba si el cliente encuentra algún servicio en particular con el nombre que proporcionó. Este emparejamiento es sintáctico, ya que, tal y como está definida la propiedad *serviceName* en la subontología Profile, tiene como rango de valores el *datatype String* del *XML Schema*, con el que solo se pueden hacer comparaciones sintácticas.
- ***filtro para nombre del proveedor**:* permite comprobar si el Servicio Web lo provee la empresa en la que está interesado el cliente. Es un parámetro como el anterior, es decir, solo se puede hacer un emparejamiento sintáctico debido a que esta propiedad de Profile también está definida como un *XML Schema String*.

Hay que destacar que los anteriores parámetros no funcionales van a incidir en el peso de distinta manera. Por ejemplo se ha querido premiar a aquellos parámetros de tipo semántico, de tal forma que si su comparación es positiva cada uno de ellos acumulará 5 puntos en el cómputo de peso relativo a este tipo de parámetros, mientras que si tratamos con los emparejamientos de *serviceName* y *ActorName* (sintácticos) aportan 3 y 2 puntos respectivamente.

- *Fase 4*: Teniendo en cuenta los pesos anteriormente calculados, ejecutar el proceso de ordenación para insertar en el lugar correcto de la lista de servicios resultado de la consulta. El servicio que encabeza la lista será aquél que se considere el más óptimo.

Las parejas obtenidas deberán ser ordenadas en función de su peso. La inserción de parejas en la lista ordenada se realiza cada vez que se construye una pareja, de tal forma que se va comparando el peso actual de la pareja recién encontrada con los pesos de cada una de las parejas ya almacenadas.

El mecanismo de inserción ordenada consiste en tomar la pareja recién encontrada y comparar su peso con el de la primera de las parejas que aparece en la lista y que será aquélla que hasta el momento poseía un mejor peso.

El hecho de que ciertos parámetros tengan mayor relevancia que otros para la búsqueda del servicio más adecuado, obliga a que la variable peso no sea manejada como algo general sino que ésta sea utilizada independientemente para cada tipo de emparejamiento. De tal forma que se acumulan pesos para salidas, entradas o parámetros no funcionales. En este último caso se considera un único valor resultado de la distinta aportación de la similitud entre parámetros dependiendo de si ésta es de tipo semántico o sintáctico.

La ordenación propuesta da más importancia al peso de las salidas, igual que en el método *sortRule()* de [13], ya que lo más importante es que el cliente obtenga lo que quiere, que debe ser lo que le proporciona el proveedor mediante las salidas (*outputs*) del servicio. A continuación, no se considera el peso de las entradas sino que se compara antes la suma de los pesos obtenidos con los parámetros de tipo no funcional que son: la proximidad en función de la región o radio geográfico, calidad de servicio, y coincidencia sintáctica del nombre del servicio y del nombre del proveedor (ver Figura 2).

Posteriormente, se compara el peso de las entradas, ya que éstos se consideran parámetros menos importantes para poder encontrar el Servicio Web que aporte lo que el cliente busca, son solamente valores de entrada antes de ejecutar el servicio y obtener de esta manera el beneficio, producto o información que el cliente espera en realidad, es decir, las salidas.

Por último, para deshacer un posible empate en este punto, se considerará el número de veces que se ha obtenido el grado de similitud *FALLO*. Debemos de tener en cuenta, que el hecho de obtener un fallo no significa el descarte de un determinado servicio, ya que en realidad, el fallo será obtenido en la búsqueda de una pareja para un parámetro dado, pero no para todos los miembros del conjunto. Por lo que en el cómputo global para ese conjunto de parámetros su valor podrá no ser nulo. De esta forma, habrá situaciones en las que para un conjunto de parámetros de la petición de un servicio, solo unos pocos consigan emparejarse con los parámetros de un determinado servicio ofertado, mientras que el número de parejas entre el conjunto de la petición y

el conjunto de parámetros de otro servicio ofertado podría ser mayor, y sin embargo obtener el mismo cómputo global, ya que se considerará la suma de todos, y en ella habrá distinta ponderación dependiendo del grado de similitud obtenido. De esta forma se ha querido premiar el hecho de que el peso esté más repartido entre todos los parámetros y no esté concentrado en unos pocos.

El proceso de **ordenación** se inicia con la comparación entre la pareja recién encontrada y la de la cabeza de la lista. Si el resultado es *false*, querrá decir que el peso no es mejor, por lo que no se habrá encontrado una mejor solución, y por tanto deberá compararse con el resto de la lista e insertarse en el lugar adecuado. En este caso, no se habrá conseguido encontrar una solución mejor, pero el registro de esta nueva pareja servirá para realizar un sistema tolerante a fallos, ya que en el caso de un fallo en el uso del servicio obtenido como mejor solución, se podrá recurrir al uso de información provista por el elemento siguiente de la lista.

```
ORDENACION(peso1,peso2)
{
      si peso1.pesosalidas > peso2.pesosalidas
            devuelve true
      sino si peso1.pesosalidas < peso2.pesosalidas
            devuelve false
      si peso1.sumaotrosparametros > peso2.sumaotrosparametros
            devuelve true
      sino
            si peso1.pesoentradas > peso2.pesoentradas
                        devuelve true
            sino si peso1.pesoentradas <= peso2.pesoentradas
                        devuelve false
            sino si peso1.numfallos < peso2.numfallos
                        devuelve true
            sino
                        devuelve false
}
```

Fig. 2 Proceso de ordenación

3.2. Descripción del proceso completo y diagrama de actividades

El emparejador actúa como un consumidor en un esquema básico productor/consumidor. Permanece dormido en *Match* (Figura 3) mientras no detecte que se haya insertado un elemento nuevo en la lista de URIs de peticiones. Cuando esto suceda, comienza el proceso de emparejamiento que se inicia con la extracción de la lista de peticiones (URIs), de aquélla que encabece la lista, el *matchmaker* o emparejador accede al *Profile* del cliente y extrae mediante un *parser*, los conceptos que definen cada uno de los parámetros. Una vez extraídos tanto los parámetros funcionales como los no funcionales, comienza el proceso de emparejamiento (Match) que se describe a continuación. (Ver Figura 3).

El proceso consiste en establecer todas las posibles parejas petición-anuncio entre la petición cursada por el cliente y cada uno de los anuncios tomados de la lista de perfiles de Servicios Web que resultó del filtrado por el parámetro categoría de servicio. Cada vez que se establece una pareja, se le aplica el método *match* (figura 4) que consiste en averiguar cada uno de los pesos parciales relativos a los parámetros de salidas, entradas y parámetros no funcionales, que nos indiquen el grado de similitud semántica entre anuncio y petición. Tanto en la comparación de las salidas como en las entradas (parámetros funcionales) se van comparando cada uno de los parámetros de la petición con los parámetros de su pareja anunciante, haciendo uso de los diferentes grados de similitud.

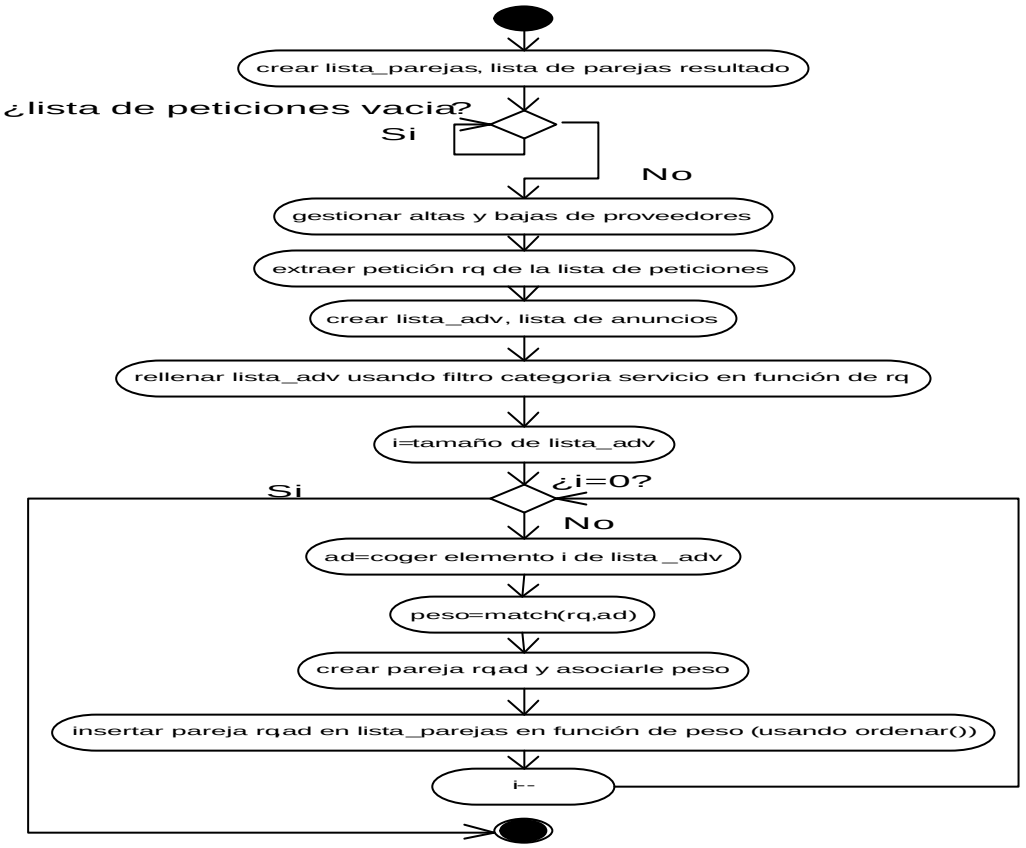


Fig. 3. Diagrama de actividades del método Match

El proceso de cálculo de pesos se realiza de la siguiente forma: (Ver Figuras 4 y 5)

Partiendo de que el sistema ha cargado inicialmente en un repositorio la ontología de conceptos asociada a la categoría de servicio definida en la petición y en los Servicios Web y una vez obtenido un *output* del anuncio y un *output* de la petición, se averigua cuál es el grado de emparejamiento, para lo cual se sigue el siguiente proceso:

- Ejecución de cada una de las consultas (*DegreeOfMatch*, figura 1), de mayor a menor grado, esperando a que alguna de las consultas devuelva resultado positivo, o, si no se consigue resultados en las consultas, se llegará al grado de emparejamiento "*Fail*" que representará que no hay emparejamiento entre los parámetros dados. Cada uno de los grados se ha cuantificado de tal forma que el grado obtenido se suma a la variable que mantiene la suma de todos los grados de similitud de los *outputs*.

Al terminar ese proceso para todos los *outputs* definidos por el usuario, y emparejados con todos los *outputs* de los Servicios Web mantenidos en el repositorio, los valores obtenidos como resultado son almacenados en la variable **peso** (*peso.salidas*).

En este punto y como mejora de anteriores propuestas, se comprueba si el peso obtenido tiene valor nulo en cuyo caso se abandona el proceso comparativo con el resto de parámetros. Si no fuera así se realizará el proceso anterior pero esta vez para los parámetros funcionales de entradas si los hubiera. El valor resultante será almacenado en *peso.entradas.*

Por último, una vez ya emparejados todos los Servicios Web en sus parámetros funcionales de entrada y salida, el siguiente paso es comprobar si estos servicios tienen definidos los parámetros no funcionales que haya especificado el usuario en su petición.

Mediante la implementación de consultas se comparan los parámetros no funcionales de radio geográfico (método *RadioGeoMatch*), calidad del servicio (método *CalidadServicioMatch*) ambos con una aportación de 5 y nombre de servicio (método *NombreServMatch*) y proveedor (método *NombreProvMatch*) con valores de 3 y 2 respectivamente. El resultado global de estas comparaciones es almacenado en la variable *peso.otros.* Al final, cada pareja tendrá asociado un peso que es el que se tomará como medida de idoneidad para la petición cursada.

A continuación y tal como ya hemos comentado, se inserta esta pareja de forma ordenada, haciendo uso del método *ordenacion()* (figura 2), en la lista de parejas resultado. Cada elemento de la lista contendrá tres valores que se corresponden con las URI´s tanto de la petición como del anuncio, así como el peso de la pareja.

4 Implementación

El algoritmo fue implementado dentro de un sistema multiagente de descubrimiento de Servicios Web de información de tráfico, y fue desarrollado en el grupo Lisitt (Laboratorio Integrado de Sistemas Inteligentes y Tecnologías de la Información en Tráfico) del Instituto de Robótica de la Universidad de Valencia. Para su desarrollo se utilizaron los siguientes componentes:

- Como repositorio se utilizó Sesame [2], debido a que permite añadir y eliminar información escrita en RDF en los repositorios, y puede almacenar esta información en cualquier base de datos. Sesame soporta como lenguajes de consulta a RQL, RDQL y SeRQL (Sesame RDF Query Language) [16], para acceder al conocimiento. Permite la interoperabilidad con un razonador de lógica descriptiva, el cual debe ser otro de los participantes en cualquier sistema emparejador.

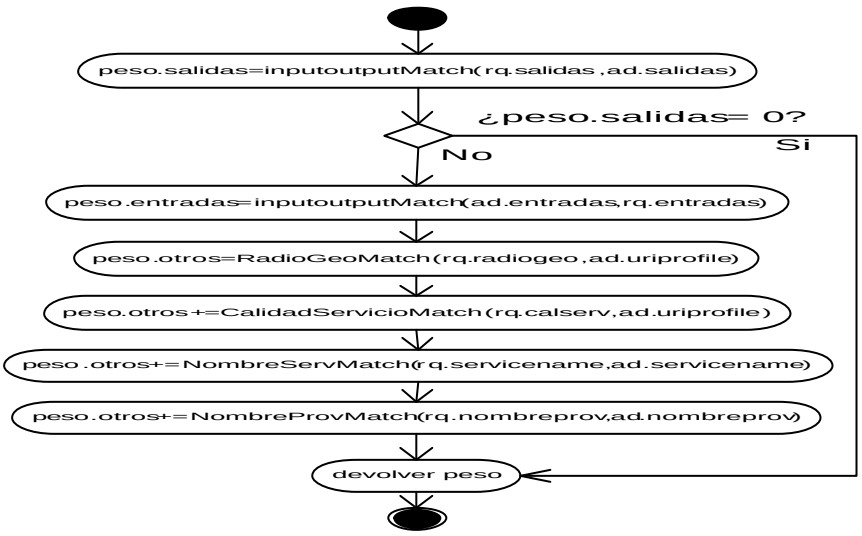


Fig. 4. Cálculo de pesos del método match

- Como razonador se utilizó BOR [3], el cual está basado en descripciones lógicas y tiene soporte para inferencias sobre instancias y sobre conceptos. Este razonador puede ser usado tanto con ontologías escritas en DAM+OIL (con algunas restricciones), y con ontologías escritas en la especificación OWL Lite. Además se puede incorporar a la aplicación Sesame, para dar soporte de ontologías DAM+OIL y poder realizar razonamiento y todas las funciones descritas en la información almacenada en los repositorios.
- Para dar soporte a DAML+OIL se utilizó el plugin de Sesame llamado SeBOR (Sesame+BOR) [14] que acepta modelos de datos semánticos de DAM+OIL en Sesame.
- El sistema de agentes que integra los diferentes componentes fue desarrollado utilizando la metodología FIPA y se implementó en lenguaje Java.

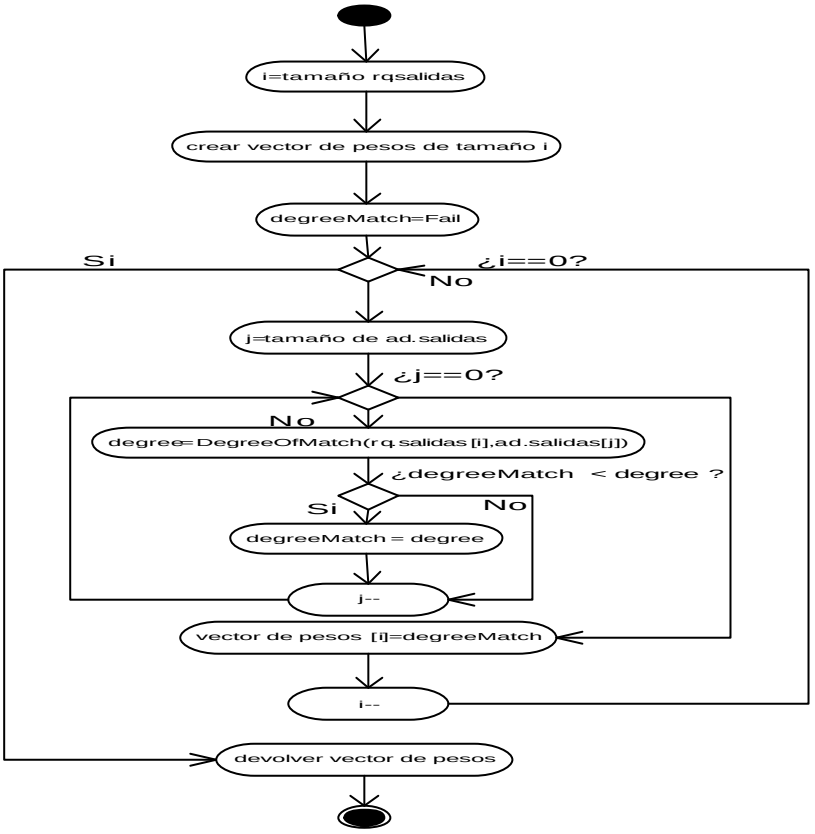


Fig. 5 Diagrama de actividades del método InputOutputMatch(), para cálculo de "peso.salidas"

4.1 Consultas para los grados de similitud en parámetros funcionales.

En las fases de diseño y posteriormente implementación se especificaron las consultas necesarias para emparejar cada uno de los grados de similitud en la fase de emparejamiento de parámetros funcionales, así como aquéllas utilizadas para comprobar el emparejamiento de los parámetros no funcionales que se emplean en el algoritmo.

A partir del estudio realizado de los lenguajes disponibles se utilizó SeRQL, debido a que era el que mejor se integraba con el razonador y repositorio seleccionado.

Dentro de las consultas implementadas para obtención de grados de similitud destaca la siguiente:

***ChermanoP*:** Esta consulta consiste en tomar las restricciones del concepto del cliente, para después navegar por las restricciones de los hermanos de esta clase, para encontrar restricciones que emparejen. En la Figura 6 puede ser observada la relación de este tipo y en la Figura 7 su implementación.

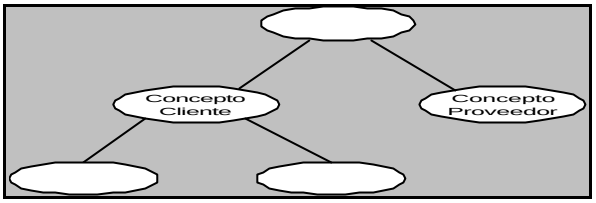


Fig. 6. Consulta por conceptos hermanos

```
Select distinct X, Y
From {X} <daml:subClassOf> {Padre},
     {Y} <daml:subClassOf> {Padre},
     {X} <rdfs:subClassOf> {Rest},
     {Y} <rdfs:subClassOf> {Rest2},
     {Rest} <rdf:type> {<daml:Restriction>},
     {Rest2} <rdf:type> {<daml:Restriction>},
     {Rest} <daml:onProperty> {Prop1},
     {Rest2} <daml:onProperty> {Prop2},
     {Rest} Z {A},
     {Rest2} Z {B}
Where X = <URI del concepto semántico del proveedor>
and Y = <URI del concepto semántico del cliente>
and Prop1=Prop2
and A = B
```

Fig. 7. Implementación de la consulta por conceptos hermanos

En el caso de que se obtenga como respuesta las URIs proporcionadas, se puede concluir que las clases son hermanas y coinciden en alguna de las restricciones.

4.2 Consultas para parámetros no funcionales

En relación con los parámetros no funcionales se implementaron consultas relacionadas con categoría de servicio, radio geográfico, calidad de servicio, nombre de servicio y nombre de proveedor. Por ejemplo, la consulta de categoría de servicio devuelve una lista de todas las URI's de los perfiles de Servicios Web que tienen como categoría de servicio la misma que pide el cliente, descartando así muchos de los perfiles que mantiene el repositorio. Esta lista de URI's de Servicios Web corresponde a los servicios que pasan a las siguientes fases del emparejamiento, liberando así al sistema de la carga de tener que aplicar el resto de filtros a todos los perfiles mantenidos en el repositorio y que no son relevantes para lo que busca el cliente, ahorrando mucho tiempo de proceso al sistema.

5 Pruebas funcionales

Se han realizado diferentes pruebas en las que se ha comprobado la mejor eficiencia del algoritmo de emparejamiento propuesto frente al algoritmo de Paolucci et al[4]. En estas pruebas se mostrará que, gracias a la mejor utilización de los parámetros de la clase *Profile* de la ontología de OWL-S / DAML-S, el algoritmo propuesto ofrece mejores resultados, en cuanto a precisión en las búsquedas de servicios, que el algoritmo proporcionado por Paolucci et al., y a su vez, los tiempos

[4] Implementación del pseudocódigo publicado en [13].

de respuesta no sufren cambios de consideración salvo en situaciones extremas (casos peor y mejor). Para realizar las pruebas se construyeron diferentes perfiles de servicios muy parecidos entre sí, y ante diferentes peticiones del cliente se estudiaron los distintos resultados dados por los dos algoritmos tras su ejecución.

En la Figura 8, se puede observar dónde se encuentran localizados en la taxonomía los principales conceptos utilizados para especificar los distintos parámetros de los servicios.

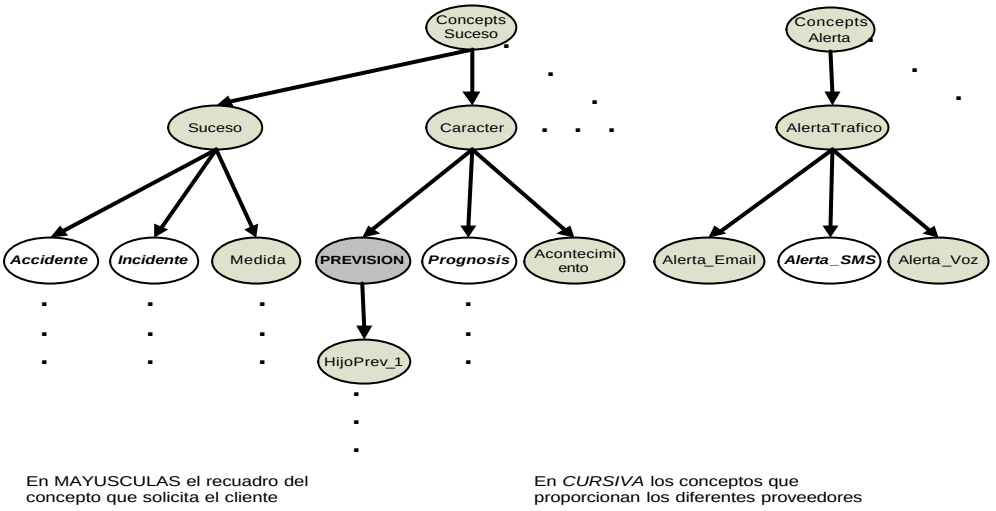


Fig. 8 Jerarquía de conceptos.

5.1 Definición de los casos de prueba.

Para comparar los dos algoritmos, a continuación se explica en qué ha consistido uno de los casos de prueba denominado: ***"Relaciones de parentesco fraternales".***

Paolucci et al. no distinguen el grado *fraternal*, por lo que las pruebas han ido dirigidas a comprobar que aunque el servicio que más se asemeja al buscado posee parámetros que pertenecen a esta relación, el algoritmo estudiado será incapaz de obtener un resultado y por tanto, devolver un perfil correspondiente a un servicio. Se comprobará que el algoritmo propuesto sí que localiza un proveedor de un concepto hermano al solicitado por el cliente. En la tabla 1 se detallan los perfiles de servicios utilizados, especificando sus parámetros mediante conceptos pertenecientes a la jerarquía establecida (Figura 8)[5]. En dicha jerarquía es posible distinguir los diferentes tipos de sucesos o eventos de tráfico, clasificados en Accidentes, Incidentes y Medidas adoptadas por la entidad competente, así como también el carácter de estos: Previsiones de tráfico, Prognosis y Acontecimientos[6]. También aparecen de forma explícita los tres tipos de servicios de alertas o notificaciones principales (por correo electrónico, SMS y uso de la voz por vía telefónica).

InfovozProfile	TraficoCatProfile
Inputs: Usuario: *XMLSchema.xsd#string* Password: *XMLSchema.xsd#string*	**Inputs:** Incidencia: *Sucesos.daml#Tipo_Incidente* Ordenación: *TraficoCatProcess.daml#Orden*

[5] Por criterios de simplicidad y claridad, no han sido expuestos todos los conceptos y jerarquías utilizadas en el emparejamiento de servicios, por ejemplo, las jerarquías de conceptos de topónimos (Geografía) o Vías.

[6] Previsiones de tráfico son sucesos desconocidos por anticipado, pero que posiblemente ocurrirán con un alto grado de probabilidad. Prognosis son sucesos conocidos por anticipado que se pueden planificar. Acontecimientos son sucesos materializados de forma no prevista.

Carretera: *Vias.daml#Via* Kminicio: *XMLSchema.xsd#decimal* Kmfin: *XMLSchema.xsd#decimal* Sentido: *Relaciones_sucesos.daml#Sentido* Dias: *Time.daml#DayofWeek* Hora_ini: *Time.daml#Time* Hora_fin: *Time.daml#Time* **Outputs:** Confirmacion: *XMLSchema.xsd#String* Alerta: *Cruta2.daml#Alerta_SMS* **Radio Geográfico:** *Geografia.daml#Espanya* **Calidad de Servicio:** *Concepts.daml#Excelente*	Demarcacion*: Geografia.daml#Provincia* Comarca: *Geografia.daml#Comarca* **Outputs:***Incidenciaout: Sucesos.daml#Incidente* **Radio Geográfico:** *Geografia.daml#Catalunya* **Calidad de Servicio:** *Concepts.daml#Excelente*
TraficoProfile **Inputs:** Incidencia: *Sucesos.daml#Tipo_Incidente* Provincia: *Geografia.daml#Provincia* Comunidades: *Geografia.daml#Comunidad_Autonoma* MostrarInc: *Concepts.daml#MostrarType* **Outputs:** Incidencia: *Sucesos.daml#Incidente* **Radio Geográfico:** *Geografia.daml#Espanya* **Calidad de Servicio:** *Concepts.daml#Excelente*	PrognosisProfile **Inputs:** Dia: *Time.daml#Date* **Outputs:** Prognosis: *Sucesos.daml#Prognosis* **Radio Geográfico:** *Geografia.daml#Espanya* **Calidad de Servicio:** *Concepts.daml#Bueno*
FechaAccidenteProfile **Inputs:** Fecha: *Time.daml#Date* **Outputs:** Salida: *Sucesos.daml#Accidente* **Radio Geográfico:** *Geografia.daml#Espanya* **Calidad de Servicio:** *Concepts.daml#Bueno*	

Tabla 1: Perfiles de servicios de proveedores usados en la prueba.

En la tabla 2 se puede observar el escenario propuesto para este caso de prueba (perfil requerido por el cliente y perfiles de los servicios ofertados por los proveedores), así como los resultados obtenidos tras la ejecución de los algoritmos.

CLIENTE	PROVEEDORES	PUNTUACIÓN (O/I/Otros)	
		Propuesto	Paolucci
ClientePrevision Profile **INPUTS:** Dia: *Time.daml#Date* **OUTPUTS:**Prevision: *Sucesos.daml#Prevision* **Radio Geográfico:** *Geografia.daml#Espanya* **Calidad de Servicio:** *Concepts.daml#Bueno*	InfovozProfile	0/0	0/0
	TraficoCatProfile	0/0	0/0
	TraficoProfile	0/0	0/0
	PrognosisProfile	**1/6**	0/0
	FechaAccidenteProfile	0/6	**0/6**

Tabla 2: Resultados escenario

Al no distinguir Paolucci et al. el grado fraternal o de hermanos, puede ocurrir que lo que solicite el cliente no se encuentre disponible mediante la búsqueda por antecedentes ni predecesores del concepto, por lo que en estos casos, es conveniente tener en cuenta en las taxonomías de conceptos, el grado de similitud entre nodos hermanos. En este caso de prueba, se observa su valor. Mientras que en nuestra propuesta el sistema ha dado como resultado el perfil correspondiente a *PrognosisProfil*e en el de Paolucci se elige el correspondiente a *FechaAccidenteProfile*.

5.2 Comparativa de los tiempos de respuesta.

Al igual que en las pruebas anteriores, hemos utilizado 2 prototipos, uno de ellos basado en nuestra propuesta de algoritmo de emparejamiento y el otro siguiendo el pseudocódigo propuesto en [13]. Las pruebas se han realizado variando el número de perfiles de servicios anunciados (1, 3, 5, 8, 10, 15, 20, 25, 30 y 33). El tiempo obtenido está mostrado en milisegundos.

Caso 1: Todos los anuncios disponibles pertenecen a la misma categoría de servicio.
Éste es el peor caso para el algoritmo propuesto. Esto es debido a que todos los perfiles de proveedores de servicios insertados en el repositorio pertenecen a la misma categoría de servicio que el servicio buscado por el cliente. En este caso el filtro de categoría de servicio no descarta perfiles, por lo que el emparejamiento se realiza en ambos prototipos con el mismo número de perfiles. Como nuestro método para calcular el grado de emparejamiento es más costoso debido a que hace más comparaciones, el prototipo basado en el algoritmo de Paolucci obtiene el resultado del emparejamiento en menos tiempo. Véase el gráfico de la figura 9.

Caso 2: El 20% de los anuncios pertenecen a la misma categoría de servicio buscada por el cliente.
Este caso muestra una situación que se aproxima más a la realidad. En ella no todos los perfiles del repositorio pertenecen a la categoría de servicio a la que pertenece el buscado por el cliente. En este caso hay un 20% de los perfiles que pertenecen a la misma categoría. Se puede observar mediante el gráfico de la figura 10 como nuestra propuesta obtiene tiempos muy similares a la de Paolucci et al.

Caso 3: Sólo hay 1 servicio que pertenezca a la categoría de servicio buscada por el cliente.
Éste es el mejor caso para nuestra propuesta, ya que sólo hay un perfil de servicio en el repositorio que coincida con la categoría buscada por el cliente. Mientras nuestra propuesta sólo hace el emparejamiento con este perfil el otro algoritmo debe comparar con todos los disponibles, por lo que nuestra propuesta obtienes tiempos considerablemente mejores. Obsérvese el gráfico de la Figura 11.

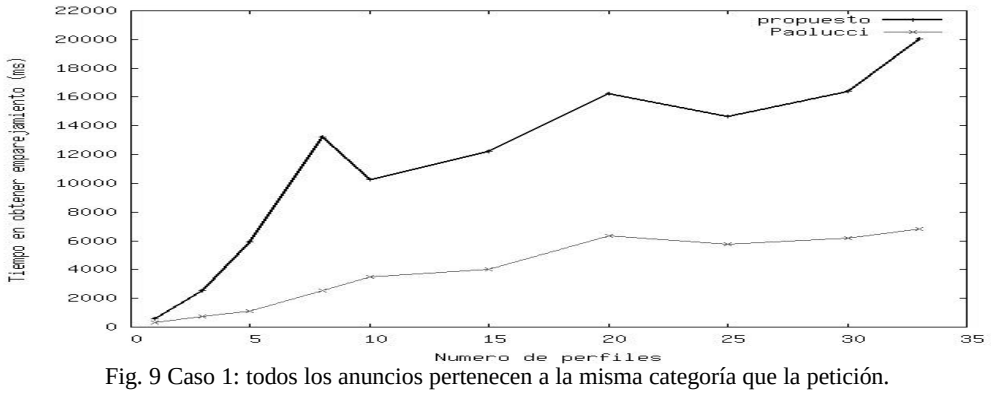


Fig. 9 Caso 1: todos los anuncios pertenecen a la misma categoría que la petición.

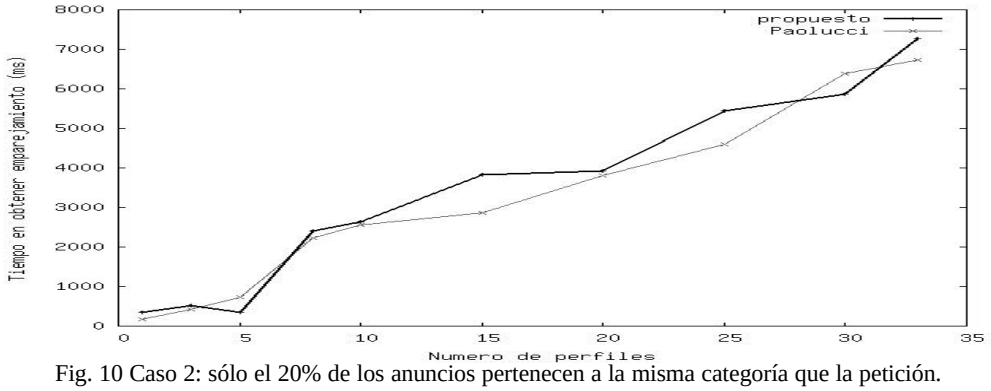


Fig. 10 Caso 2: sólo el 20% de los anuncios pertenecen a la misma categoría que la petición.

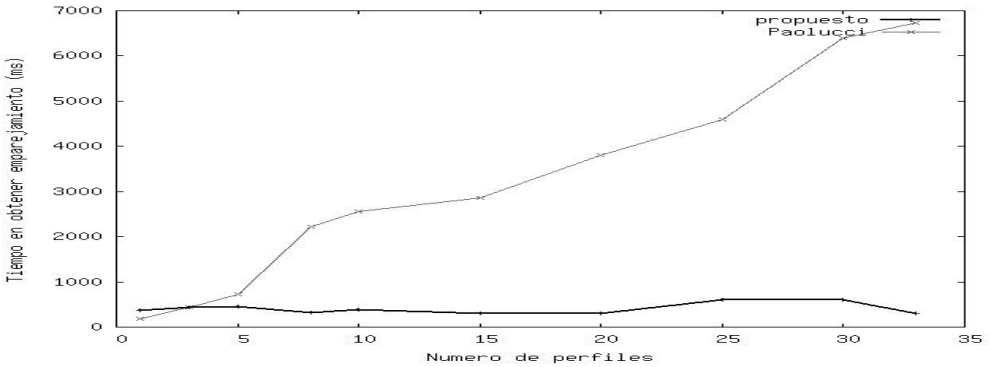


Fig. 11 Caso 3: sólo uno de los anuncios pertenece a la misma categoría.

6 Análisis de Resultados

A partir de las pruebas efectuadas se pudo observar como hecho más importante, que utilizando el algoritmo fue posible encontrar servicios basados en similitudes de tipo hermano, las cuales no habían sido estudiadas anteriormente. Aunque este tipo de similitud pudo no haberse tenido en cuenta debido a que en el caso de encontrarse similitudes entre nodos hermanos, posiblemente sería recomendable desde el punto de vista de diseño crear un nodo padre intermedio que agrupase las características comunes de los hijos, nuestro algoritmo las tiene en cuenta partiendo de la premisa de que es posible que este tipo de similitudes no hayan sido consideradas por el proveedor que definió semánticamente estas relaciones, bien sea por consideraciones de rendimiento (para evitar un grado de profundidad demasiado exhaustivo) o simplemente por no haber detectado este tipo de relaciones en la fase de diseño, con lo que el descubrimiento de servicios basado en algoritmos convencionales dejaría de producir resultados que están incluidos en el rango requerido de respuesta.

Por otra parte, la elección de ontologías en las que no hay conceptos con definiciones completas (condiciones necesarias y suficientes para ser miembro de una clase) puede dar lugar a este tipo de situaciones, ya que los razonadores serán incapaces de establecer clasificaciones de manera automática. En estos casos, el peso de la construcción de una buena y apropiada ontología recaerá en el desarrollador de ésta. Por el contrario, el diseño de ontologías con conceptos definidos permitirá solventar este tipo de problema aunque incrementará notablemente el coste computacional, por lo que la decisión de incorporar este grado de similitud se ve ampliamente respaldada.

Otras decisiones como el establecimiento de filtrados anteriores al algoritmo así como el uso de otros parámetros no funcionales, mejoraron considerablemente el coste y la elección del servicio más idóneo respectivamente.

Es importante destacar ciertas mejoras introducidas en el algoritmo que también supusieron un ahorro en el coste computacional como la finalización del proceso comparativo, con un determinado anuncio, cuando no hay emparejamientos de parámetros de salida, y la posibilidad de hacer uso del resto de parejas (petición-anuncio) almacenadas tras el resultado de la búsqueda, para establecer un sistema tolerante a fallos.

Tras las pruebas realizadas, podemos afirmar que el algoritmo propuesto, al contener muchas más consultas (comparaciones) debido a la inclusión de los nuevos grados y accesos al repositorio donde están las ontologías y los perfiles, tiene un mayor coste temporal, el cual es únicamente importante en casos extremos en los que exista un número muy elevado de anuncios de proveedores que pertenezcan a la misma categoría de servicio que el de la petición del cliente. Sin embargo, en situaciones próximas a la realidad, donde aproximadamente el 20 % de los anuncios

pertenecerán a la misma categoría de servicio, el coste temporal es similar al del algoritmo de Paolucci et al. e incluso lo mejora en escenarios donde el número de anuncios pertenecientes a la categoría del de la petición es muy bajo. Por ello, podemos concluir que se ha conseguido aumentar la precisión en las búsquedas sin perjudicar el coste temporal en situaciones consideradas normales.

7 Conclusiones

En este artículo se ha presentado un resumen del estado del arte de sistemas de emparejamiento y diferentes algoritmos integrados en ellos. El principal punto de investigación han sido los diferentes grados de emparejamiento que permitieran flexibilidad, uso de parámetros no funcionales así como los diversos filtros usados, algunos de los cuales no habían sido considerados en anteriores trabajos de otros autores. La principal contribución al mundo de los Servicios Web Semánticos fue no sólo reflejar el aspecto teórico, sino también el aspecto práctico mediante la construcción de un sistema capaz de hacer uso del algoritmo de emparejamiento propuesto.

En esta propuesta se decidió ampliar el rango de grados de similitud de anteriores propuestas, porque se consideraba que alguno de los grados de similitud definidos eran demasiado generales. Éste era el caso del grado "*exact*" definido por Paolucci et al., que incluye los casos en los que el concepto definido por el cliente en su petición es exactamente el mismo o es subclase del concepto que definió el proveedor en su perfil. En nuestro algoritmo distinguimos dos subgrados, uno, el de mayor valor, que será aquel en el que tanto proveedor y cliente utilizan el mismo concepto para definir un parámetro funcional, y un segundo subgrado que será aquél en el que el concepto semántico descrito por el cliente es subclase directa del que define el proveedor. Adicionalmente se ha descrito un nuevo grado de similitud no considerado por Paolucci et al, consistente en la relación entre "hermanos" que se puede dar con los dos conceptos dados. Este grado consiste en que dados dos conceptos, se les considera hermanos si ambos son subclase directa de otro concepto y poseen una restricción común sobre alguna de sus propiedades.

El algoritmo fue implementado como un componente dentro de un sistema multiagente para publicación y descubrimiento de servicios. Es independiente del lenguaje de especificación de ontologías que se utilice para describir, tanto las descripciones de los Servicios Web como las diferentes ontologías utilizadas para calcular el grado de similitud que se tenga. Por el momento, el sistema funciona únicamente con el lenguaje de especificación DAML+OIL para las ontologías de conceptos y DAML-S para la descripción de los Servicios Web. Se decidió no utilizar el lenguaje OWL como lenguaje de marcado semántico debido a que en el momento de iniciar la implementación del prototipo este lenguaje aún no era soportado por el razonador y repositorio seleccionados. Sin embargo, se han creado descripciones de los servicios tanto en DAML-S 0.9 como en OWL-S 1.0 porque se prevé que próximamente Sesame dé soporte a OWL, por lo que este sistema podrá migrarse aplicando el mismo algoritmo de emparejamiento desarrollado.

Referencias

[1] C.Abela; M.Montebello. DAML enabled Web Services and Agents in the Semantic Web. En: WS--RSD'02, Erfurt Germany, October 2002.

[2] J. Broekstra; A. Kampman and F. Van Harmelen. Sesame: a Generic Architecture for Storing and Querying RDF and RDF Schema. En: 1st Int. Semantic Web Conference, Italy, 06- 2002.

[3] K. Simov and S. Jordanov. BOR: a Pragmatic DAML+OIL Reasoner; Deliverable 40, On-To-Knowledge project, Junio 2002.

[4] A. Ankolenkar et al. The DAML Services Coalition. DAML-S: Web Service Description for the semantic Web. En: The First International Semantic Web Conference (ISWC), 2002.

[5] Matchmaker by CMU. http://www-2.cs.cmu.edu/~softagents/daml_Mmaker/daml-s_matchmaker.htm Última visita 07-11- 2004.

[6] S. Colucci et al. Logic Based Approach to Web Services Discovery and Matchmaking. En: Modeling E-services Workshop at 5th International Conference on Electronic Commerce, (ICEC'03). Pittsburgh, October 3, 2003.
[7] J. Gonzalez-Castillo; D. Trastour and C. Bartolini. "Description Logics for MatchMaking of Services"; HP Technical Reports. October 30 th , 2001
[8] M. Klein, and A. Bernstein. Searching for Services on the Semantic Web using Process Ontologies. En: The First Semantic Web Working Symposium (SWWS-1). Stanford, 2001.
[9] L. Lei and I. Horrocks. A software Framework For Matchmaking Based on Semantic Web Technology. En: Twelfth International World Wide Conference (WWW2003), pages 331-339, ACM, 2003.
[10] D. Martin, M. Paolucci and S. McIlraith. List of Web Services. En: http://lists.w3.org/Archives/Public/www-ws/2002Mar/0009.html
[11] D. Martin. List of Web Services. En: http://lists.w3.org/Archives/Public/www-ws/2003May/0028.html
[12] T. Di Noia et al. A System for Principled Matchmaking in an Electronic Marketplace. En: WWW2003.
[13] M. Paolucci et al. "Semantic Matching of Web Services Capabilities". In The First International Semantic Web Conference (ISWC), 2002.
[14] S. Jordanov and K. Simov, BOR: a Pragmatic DAML+OIL Reasoner, System Documentation Overview, Installation, Integration, http://www.ontotext.com/BOR, 21.11.2002
[15] T. R. Payne ; M. Paolucci and K. Sycara "Advertising and Matching DAML-S Service Descriptions". En: Semantic Web Working Symposium (SWWS), 2001.
[16] B.V. Aduna, Sirma AI Ltd. Chapter 6. The SeRQL query language, rev. 1.1 from User Guide for Sesame, http://www.openrdf.org/doc/users/ch06.html., Última visita noviembre 2004.
[17] K. Sycara et al. Dynamic Service Matchmaking Among Agents in Open Information Environments. En: Journal ACM SIGMOD Record, 1999.
[18] D. Trastour; C. Bartolini and J. Gonzalez-Castillo. A Semantic Web Approach to Service Description for Matchmaking of Services. En: 1st Semantic Web Working Symposium, CA. 2001.
[19] I. Constantinescu, B. Faltings. World Wide Web Consortium Efficient Matchmaking and Directory Services. Technical Report No IC/2002/77, Noviembre, 2002.
[20] A. Zaremski and M. Wing. Signature matching: a Tool for Using Software Libraries. ACM Transactions on Software Engineering and Methodology, 1995
[21] A. Zaremski and M. Wing. Specification Matching of Software Components. ACM Transactions on Software Engineering and Methodology (TOSEM), 1997.
[22] T. R. Payne and C. Abela, List of Web Services. En: http://lists.w3.org/Archives/Public/www-ws/2002Jan/0008.html, Enero 2002.
[23] R. Ferraz; S. Labidi and B. Wanghon. "A semantic matching method for clustering traders in B2B Systems", 1er Latin American Web Congress (LA-WEB 2003), 0-7695-2058-8/03.